МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

ВІВЮРКА Назар Миколайович

Модель прогнозування дефектів програмних проєктів на основі машинного навчання / Software Project Defect Prediction Model Based on Machine Learning

спеціальність: 122 - Комп'ютерні науки
освітньо-професійна програма - Управління проектами

Кваліфікаційна робота

Виконав студент групи КНУПм-21
Вівюрка Н.М.
_____

Науковий керівник:
к.т.н., доцент, Майків І.М.
_____

Кваліфікаційну роботу
допущено до захисту:
«___» _____ 20___ р.
В.о. завідувача кафедри
_____ Н.М. Васильків

ТЕРНОПІЛЬ - 2024

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «магістр»
спеціальність: 122 – Комп'ютерні науки
освітньо-професійна програма – Управління проектами

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ М.П. Комар
«____» _____20__р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
ВІВЮРКА Назар Миколайович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Модель прогнозування дефектів програмних проєктів на основі машинного навчання / Software Project Defect Prediction Model Based on Machine Learning
керівник роботи к.т.н., доцент, Майків І.М.
затверджені наказом по університету від 12 грудня 2023 року № 753.

2. Строк подання студентом закінченої кваліфікаційної роботи 4 грудня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.

4. Основні питання, які потрібно розробити
- Проаналізувати виклики управління складністю програмного забезпечення та сучасні інструменти прогнозування дефектів.
- Провести огляд існуючих рішень для прогнозування дефектів у програмних проєктах.
- Розробити модель прогнозування дефектів між проєктами на основі машинного навчання.
- Дослідити різні класифікатори на основі машинного навчання для прогнозування дефектів.
- Запропонувати нові структурні метрики для покращення об'єктно-орієнтованих систем.
- Провести порівняльний аналіз продуктивності алгоритмів машинного навчання на основі метрик коду.
- Проаналізувати чутливість та результати запропонованої моделі в контексті існуючих досліджень..

5. Перелік графічного матеріалу у роботі
- Загальна структура моделі CPSDP.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 12 грудня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи.	до 01.01. 2024 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.03. 2024 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 20.05.2024 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 28.10. 2024 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 11.11.2024 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів.	до 25.11.2024 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту.	до 30.11.2024 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 04.12.2024 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії.	до 14.12. 2024 р.	

Студент _____ Вівюрка Н.М.
підпис

Керівник роботи _____ к.т.н., доцент, Майків І.М.
підпис

РЕЗЮМЕ

Кваліфікаційна робота на тему «Модель прогнозування дефектів програмних проєктів на основі машинного навчання» на здобуття освітнього ступеня «Магістр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Управління проектами» написана обсягом 89 сторінок і містить 9 ілюстрацій, 3 таблиць, 7 додатків та 51 використаних джерела.

Метою даної кваліфікаційної роботи є розробка моделі прогнозування дефектів програмних проєктів, яка використовує машинне навчання для підвищення точності прогнозів і забезпечення надійності програмного забезпечення.

Методи дослідження: аналіз наукової літератури, алгоритми машинного навчання для прогнозування дефектів, розробка метрик коду для покращення точності моделей.

Результати дослідження: розроблено модель прогнозування дефектів на основі алгоритмів машинного навчання, яка використовує нові метрики коду для ефективного прогнозування дефектів. Модель показала високу точність в порівнянні з традиційними методами.

Практичне значення роботи: розроблена модель може бути використана для покращення якості програмного забезпечення через точніше прогнозування дефектів на ранніх етапах розробки.

Ключові слова: ПРОГНОЗУВАННЯ ДЕФЕКТІВ, МАШИННЕ НАВЧАННЯ, МЕТРИКИ КОДУ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, МОДЕЛІ ПРОГНОЗУВАННЯ.

ABSTRACT

Qualification Thesis titled "Software Project Defect Prediction Model Based on Machine Learning" for the award of the educational degree "Master" in the specialty 122 "Computer Science," educational program "Project Management," is written in 89 pages and includes 9 illustrations, 3 tables, 7 appendices, and 51 references.

The goal of this qualification thesis is to develop a software defect prediction model using machine learning to improve prediction accuracy and ensure the reliability of software systems.

Research methods: analysis of scientific literature, machine learning algorithms for defect prediction, development of code metrics to improve model accuracy.

Research results: A defect prediction model has been developed based on machine learning algorithms, which uses new code metrics for effective defect prediction. The model showed high accuracy compared to traditional methods.

Practical significance of the work: The developed model can be used to improve software quality by more accurately predicting defects at the early stages of development.

Keywords: DEFECT PREDICTION, MACHINE LEARNING, CODE METRICS, SOFTWARE, PREDICTION MODELS.

ЗМІСТ

ВСТУП 8
1 Теоретичні основи прогнозування дефектів у програмних проєктах 12
1.1 Виклики управління складністю програмного забезпечення та сучасні інструменти прогнозування дефектів 12
1.2 Огляд існуючих рішень 14
1.3 Постановка задачі 18
Висновки до розділу 1 21
2 Модель прогнозування дефектів між проєктами 23
2.1 Модель прогнозування дефектів між проєктами 23
2.2 Класифікатори на основі машинного навчання 37
2.3 Запропоновані структурні метрики для покращення об'єктно-орієнтованих систем 40
Висновки до розділу 2 43
3 Практичне використання отриманих наукових результатів 44
3.1 Порівняння продуктивності алгоритмів машинного навчання для прогнозування дефектів на основі метрик коду 44
3.2 Аналіз чутливості запропонованої моделі 53
3.3 Аналіз результатів запропонованої моделі у контексті існуючих досліджень 54
Висновки до розділу 3 58
ВИСНОВКИ 59
СПИСОК ВИКОРИСТАНих джерел 61
Додаток А Підсумок існуючих досліджень 67
Додаток Б Вплив розміру ансамблю на значення auc (аналіз чутливості) 69
Додаток В Порівняння складності моделей 70
Додаток Г порівнянн метрик точності 72

Додаток Д Порівняння складності запропонованої моделі з існуючими моделями..
..... 76
Додаток Ж Набір метрик..... 77
Додаток З Апробація отриманих результатів 79

ВСТУП

Прогнозування дефектів програмного забезпечення є критично важливим для забезпечення його якості та надійності, оскільки виявлення помилок на ранніх етапах розробки допомагає знизити витрати на виправлення дефектів і зменшити час на випуск продукту. Зростання масштабів сучасних програмних систем і їхня складність роблять ручні методи тестування та аналізу неефективними. Це створює необхідність у автоматизованих рішеннях, таких як моделі прогнозування дефектів на основі машинного навчання. Актуальність дослідження полягає в тому, що існуючі підходи до прогнозування часто використовують застарілі метрики коду, які не здатні повною мірою відобразити всі аспекти об'єктно-орієнтованого програмування, що призводить до неточних прогнозів і збільшення кількості невиявлених дефектів.

У сучасному середовищі програмування дедалі більше використовуються об'єктно-орієнтовані мови, такі як Java, C++, Python, де важливі такі аспекти, як успадкування, поліморфізм та згуртованість класів. Однак багато існуючих моделей прогнозування дефектів не враховують цих характеристик достатньо ефективно, що знижує точність їхніх прогнозів. Враховуючи це, розвиток нових метрик, які можуть повніше охоплювати ці особливості, є нагальною необхідністю для підвищення якості програмного забезпечення. Запропоновані в дослідженні метрики, такі як ETCC, ECAC, ELCOCC, орієнтовані на вирішення цієї проблеми та здатні покращити якість моделей прогнозування.

Крім того, важливим аспектом є актуальність міжпроєктного прогнозування дефектів (CPSDP), коли для навчання моделей використовуються дані з різних проєктів. Це особливо корисно у випадках, коли немає достатньо даних для конкретного проєкту або ці дані є недостатньо репрезентативними. Міжпроєктне прогнозування дає можливість покращити моделі за рахунок використання знань із подібних проєктів, що робить ці моделі більш універсальними і ефективними в

реальних умовах. У зв'язку з цим розробка моделей на основі машинного навчання для CPSDP є необхідною для сучасної індустрії програмного забезпечення.

Нарешті, швидкість і ефективність роботи таких моделей стають дедалі важливішими, враховуючи обсяги сучасних програмних систем. Існуючі моделі на основі глибокого навчання, такі як CNN і TCNN, потребують значних обчислювальних ресурсів і часу на навчання, що обмежує їхнє використання в реальних умовах. Запропонована модель на основі XGBoost та нових структурних метрик є не тільки точнішою, але й значно легшою та швидшою в роботі, що робить її актуальною для впровадження у великих проєктах, де важливі як точність, так і швидкість прогнозування.

Метою даної роботи є розробка моделі прогнозування дефектів програмних проєктів на основі алгоритмів машинного навчання, яка дозволить покращити точність і ефективність виявлення потенційних дефектів у програмному забезпеченні на етапі його розробки та підтримки.

Задачі, які необхідно вирішити для досягнення поставленої мети, включають:

1. Проаналізувати виклики управління складністю програмного забезпечення та сучасні інструменти прогнозування дефектів.

2. Провести огляд існуючих рішень для прогнозування дефектів у програмних проєктах.

3. Розробити модель прогнозування дефектів між проєктами на основі машинного навчання.

4. Дослідити різні класифікатори на основі машинного навчання для прогнозування дефектів.

5. Запропонувати нові структурні метрики для покращення об'єктно-орієнтованих систем.

6. Провести порівняльний аналіз продуктивності алгоритмів машинного навчання на основі метрик коду.

7. Проаналізувати чутливість та результати запропонованої моделі в контексті існуючих досліджень.

Об'єктом дослідження є процеси прогнозування дефектів у програмних проєктах на основі машинного навчання.

Предметом дослідження є методи та моделі машинного навчання, а також структурні метрики коду для прогнозування дефектів між програмними проєктами.

Методи дослідження включають аналіз і узагальнення літературних джерел для визначення сучасних підходів до прогнозування дефектів у програмному забезпеченні, розробку моделі на основі машинного навчання із застосуванням різних алгоритмів класифікації, таких як XGBoost, Random Forest, SVM та інших. Крім того, використовуються методи нормалізації даних і налаштування гіперпараметрів через перехресну перевірку (5-fold cross-validation). Для оцінки ефективності моделі застосовано метрики продуктивності, такі як AUC, точність, f1-score та аналіз чутливості на основі тестових наборів даних з різних проєктів.

Наукова новизна роботи полягає у розробці нової моделі прогнозування дефектів програмних проєктів між різними проєктами (CPSDP) на основі машинного навчання з використанням запропонованого набору структурних метрик коду. Вперше враховано комплексний підхід до аналізу об'єктно-орієнтованих аспектів програмного забезпечення, включаючи складність класів, зчеплення, згуртованість методів та поліморфізм. Запропоновані метрики, такі як ETCC, ECAC, ELCOCC, дозволили суттєво підвищити точність прогнозування дефектів порівняно з існуючими моделями, що підтверджено результатами тестування на реальних проєктах із відкритим кодом.

Практичне значення роботи полягає в можливості застосування розробленої моделі прогнозування дефектів програмних проєктів (CPSDP) у реальних умовах розробки програмного забезпечення, що дозволяє зменшити витрати на тестування та підвищити якість кінцевого продукту. Впровадження нових структурних метрик коду забезпечить точніше оцінювання ризиків дефектів, що, в свою чергу, допоможе командам розробників своєчасно виявляти проблеми у коді і оптимізувати процеси управління проектами. Завдяки покращеній продуктивності алгоритмів машинного навчання, використаних у моделі, організації зможуть

досягати більшої ефективності у прогнозуванні дефектів, зменшуючи ризики та забезпечуючи стабільнішу роботу програмних продуктів.

Структура та обсяг роботи. Робота складається із вступу, трьох розділів, висновків, списку використаних джерел та додатків.

Апробація результатів дослідження. Основні теоретичні положення роботи й практичні результати дослідження доповідалися й обговорювалися на V Міжнародній науковій конференції «Стратегічні напрямки розвитку науки: фактори впливу та взаємодії»,яка відбулася 11 жовтня 2024 року у місті Луцьк та VI Всеукраїнської мультидисциплінарної студентської наукової конференції «Розвиток сучасної науки: актуальні питання теорії та практики», яка відбулася 20 вересня 2024 року у місті Кропивницький, Україна.

1 ТЕОРЕТИЧНІ ОСНОВИ ПРОГНОЗУВАННЯ ДЕФЕКТІВ У ПРОГРАМНИХ ПРОЄКТАХ

1.1 Виклики управління складністю програмного забезпечення та сучасні інструменти прогнозування дефектів

Використання програмного забезпечення зростає, оскільки воно інтегрується в різні галузі, такі як торгівля, інженерія, Інтернет речей тощо. Відповідно, складність програмного забезпечення також зростає експоненціально [1, 2]. Основною метою команди з управління проєктом є розробка надійного та безпомилкового програмного забезпечення з обмеженим бюджетом та в рамках часу, виділеного компанією [3]. Однак через складність програмного забезпечення досягнення цих цілей в обмежений час та з обмеженим бюджетом стає складним, що призводить до появи модулів програмного забезпечення, схильних до дефектів [4 , 5]. Для вирішення цієї проблеми були розроблені автоматизовані інструменти для виявлення дефектних модулів програмного забезпечення [6 - 9]. Основною метою інструментів прогнозування дефектів програмного забезпечення (SDP) є ідентифікація дефектних модулів, щоб зменшити навантаження на команду тестування і зробити процес тестування швидшим, що дозволяє заощадити бюджет та скоротити час розробки проєкту [10, 11].

Дослідники зацікавлені в галузі прогнозування дефектів програмного забезпечення через переваги автоматизованого тестування, і в останні роки в цій галузі було проведено багато досліджень [7, 8, 12-14]. На основі досліджень було запропоновано два типи моделей прогнозування дефектів: прогнозування дефектів у рамках одного проєкту та прогнозування дефектів на основі інших проєктів [30]. Прогнозування дефектів на основі інших проєктів (CPSDP) є більш практичним, ніж прогнозування дефектів у рамках одного проєкту, оскільки в реальних сценаріях або дуже обмежена, або взагалі відсутня інформація про попередні версії програмного забезпечення, що розробляється [15, 16]. Тому для прогнозування

дефектів використовуємо інші проєкти, розроблені іншими компаніями або тією ж компанією, що називається прогнозуванням дефектів на основі інших проєктів (CPSDP). Ці проєкти (історичні дані) необхідні, оскільки більшість моделей прогнозування дефектів базуються на алгоритмах машинного навчання (ML), а машинному навчанню потрібні історичні дані для навчання моделі перед фактичним прогнозуванням. Однак історичні дані не можна безпосередньо подавати як вхід до алгоритмів машинного навчання. Нам потрібно виділити особливості коду, які називаються метриками коду в інженерії програмного забезпечення. Ці метрики коду кількісно визначають об'єктно-орієнтовану структуру проєкту на основі таких характеристик, як складність класів, зв'язність класів через виклики методів, відсутність згуртованості, зв'язок по успадкуванню, поліморфізм і абстракція, які використовуються для вимірювання якості коду.

Ці структурні метрики коду використовуються як вхідні дані для навчання моделей CPSDP на основі ML, і продуктивність моделей CPSDP на основі ML залежить від якості цих метрик. Тому необхідно виділяти якісні метрики вихідного коду, базуючись на різних структурних аспектах мов програмування. Дослідники запропонували багато метрик вихідного коду на основі об'єктно-орієнтованої структури мов програмування, таких як метрики CK [17], метрики MOOD [18], метрики QMOOD [19] та багато інших. Однак є потреба в удосконаленні метрик вихідного коду для більш ефективного відображення структурних особливостей мов програмування, оскільки існуючі структурні метрики коду базуються на тривіальних підрахунках характеристик і є недостатніми для глибокого аналізу програмного забезпечення для оцінки якості коду. Тому в цьому дослідженні пропонується новий набір метрик вихідного коду, розроблений для більш ефективного відображення структурних особливостей коду. Оскільки новий набір метрик вихідного коду, також необхідно визначити найбільш підходящий алгоритм ML для розробки хорошої моделі CPSDP на основі запропонованих структурних метрик вихідного коду. Загалом, необхідно розробити кращу модель CPSDP,

використовуючи поліпшені метрики вихідного коду та вибір найбільш підходящого алгоритму ML для цих метрик вихідного коду.

Це дослідження пропонує покращену модель CPSDP на основі ML, використовуючи новий набір метрик, виділений на основі структурних особливостей об'єктно-орієнтованого програмного забезпечення. Запропонований набір метрик містить шість нових метрик коду для оцінки якості коду на основі структурних особливостей мов програмування, таких як складність класів, зв'язність класів через виклики методів, відсутність згуртованості, зв'язок по успадкуванню, абстракція і поліморфізм. Ці метрики коду використовуються як вхідні характеристики для нашої запропонованої моделі CPSDP на основі ML. Запропоновано нову процедуру, засновану на парному t-тесті, для вибору проєктів для підготовки навчального набору даних. Нарешті, для запропонованої моделі CPSDP обирається найбільш підходящий алгоритм ML на основі продуктивності різних алгоритмів ML. Шість існуючих показників продуктивності, таких як AUC, точність, precision, false omission rate, recall та f1-score, а також новий показник продуктивності під назвою BBAA (баланс між AUC і точністю), запропонований у цій статті, використовуються для порівняння продуктивності алгоритмів ML.

1.2 Огляд існуючих рішень

Моделі прогнозування дефектів програмного забезпечення (SDP) використовуються для виявлення дефектних модулів програмного забезпечення, яке розробляється, з метою зменшення витрат і зусиль команди тестування. Більшість сучасних моделей SDP базуються на алгоритмах машинного навчання, які використовують метрики якості коду як вхідні характеристики для навчання моделі. Було запропоновано велику кількість метрик, заснованих на об'єктно-орієнтованій концепції. Однак не всі метрики необхідні для розробки моделей SDP, оскільки багато з них є надлишковими у літературі. Із усіх існуючих метрик коду в літературі було виділено 20 метрик коду, які охоплюють усі аспекти об'єктно-

орієнтованого програмування (складність класу, зв'язок між класами через виклики методів і успадкування, відсутність згуртованості, поліморфізм, абстракція та приховування), і були розроблені набори даних прогнозування дефектів для більш ніж двадцяти проєктів, що містяться в репозиторії Promise. Усі останні дослідження CPSDP базуються на цих наборах даних з репозиторію Promise. Тут представлено деякі з останніх досліджень CPSDP на основі ML, які використовують метрики коду Promise як вхідні характеристики.

Арар та ін. [36] запропонували вдосконалений алгоритм штучної колонії бджіл (ABC) для навчання нейронної мережі, чутливої до витрат, для прогнозування дефектів. Було розроблено нову функцію мети, засновану на TPR та TNR, помножених на коефіцієнти витрат, яку мінімізує запропонований варіант ABC. Запропонований підхід був протестований на п'яти проєктах NASA, і результати показують, що він ефективніший за традиційну штучну нейронну мережу для незбалансованої класифікації.

Арар та ін. [12] запропонували новий класифікатор Наївного Баєса, залежний від характеристик (FDNB), для кращого прогнозування дефектів. Після попередньої обробки даних він виводить залежність між незалежними характеристиками, щоб зробити їх залежними, і тоді для прогнозування дефектів використовується класифікатор Наївного Баєса. Результати показують, що запропонований FDNB перевершує всі інші варіанти Наївного Баєса.

Рю та ін. [9] запропонували новий класифікатор Наївного Баєса для прогнозування дефектів між проєктами (CPSDP) з множинними цілями (MONB). Було виведено три цілі: показник помилкових пропусків, ймовірність виявлення та баланс між ними для багатозадачного Наївного Баєса. Для навчання Наївного Баєса на основі цих трьох цілей використовується алгоритм гармонійного пошуку для багатозадачності, і результати показують, що запропонований MONB працює краще, ніж однозадачні моделі.

Пандей та ін. [14] продемонстрували застосування глибокої нейронної мережі на основі уваги для CPSDP. Результати моделі були порівняні з вісьмома

еталонними алгоритмами машинного навчання, і доведено, що нейронна мережа на основі уваги перевершує інші алгоритми машинного навчання.

Кабір та ін. [8] показали важливість вибору характеристик у тимчасовому навчанні для прогнозування дефектів між версіями програмного забезпечення (SDP). Вони розглядають кожний випуск програмного забезпечення як частину навчальних даних, і кожен новий випуск змінює важливість характеристик у навчанні. Тому вибір характеристик є важливим для створення нової моделі прогнозування дефектів на основі попередніх випусків програмного забезпечення. Результати показують, що ретельний вибір характеристик може зменшити зсуви концепції.

Сунь та ін. [41] запропонували новий підхід колаборативної фільтрації для вибору подібних наборів даних для навчання моделей CPSDP на основі ML, оскільки вибір навчальних проєктів є найважливішим у випадку CPSDP. Результати показують, що вибір на основі запропонованого підходу фільтрації допомагає створювати кращі моделі прогнозування дефектів, ніж у попередніх дослідженнях.

Цю та ін. [42] запропонували новий варіант згорткової нейронної мережі (CNN) під назвою переносна згорткова нейронна мережа (TCNN), яка видобуває переносні семантичні характеристики для подолання обмежень існуючих структурних та семантичних характеристик. Додано шар узгодження для перетворення CNN на TCNN, де дані вихідного та цільового проєктів впроваджуються у відтворювальний гільбертовий простір для узгодження розподілу. Експерименти на основі десяти проєктів з відкритим кодом доводять, що запропонована TCNN досягає кращих результатів, ніж моделі-еталони.

З усіх існуючих досліджень CNN, TCNN та ще шість останніх досліджень, таких як комбінований прогнозувальник дефектів (CODEP), аналіз компонентів перенесення (TCA+), гібридний підхід до реконструкції моделі (HYDRA), двофазне перенесення навчання (TPTL), TDS та гібридний індуктор ансамблевого

навчання (HIEL), були використані в цьому дослідженні для порівняння нашої запропонованої моделі CPSDP, підсумовані в таблиці A.1 (див. додаток A).

Ці шість останніх досліджень були обрані, оскільки вони використовують існуючі структурні метрики коду для прогнозування дефектів програмного забезпечення між проєктами (CPSDP), і наше дослідження має на меті показати переваги запропонованих метрик коду над існуючими тривіальними метриками, що базуються на підрахунку, для захоплення об'єктно-орієнтованих властивостей програмного забезпечення. HIEL є найкращою відомою моделлю CPSDP на основі існуючих структурних метрик коду для CPSDP. Отже, обравши ці дослідження для порівняння нашої запропонованої моделі. Однак також обрано два дослідження, засновані на семантичних характеристиках, такі як CNN і TCNN (transfer CNN), щоб зробити порівняння більш різноманітним.

Дослідження, викладені в таблиці A.1, використовують набори даних Promise, а ці набори даних створені на основі 20 існуючих метрик вихідного коду, які охоплюють всі структурні аспекти об'єктно-орієнтованого програмування, такі як складність класів, зв'язність через виклики методів та успадкування, відсутність згуртованості, абстракція, приховування даних та поліморфізм. Проте існує багато інших метрик коду, і розвиток метрик вихідного коду підсумовується в наступному абзаці.

Холстед та ін. [20] запропонували набір метрик, заснований на тривіальному підрахунку операторів і операндів виразів, а МакКейб [21] запропонував нову метрику коду, відому як цикломатична складність МакКейба, яка підраховує загальну кількість лінійно незалежних шляхів у графі потоку керування функції. Ці метрики коду базувалися на функціональному програмуванні й не могли відобразити структурні аспекти об'єктно-орієнтованого програмування. У 1994 році Чідамбер і Кемерер [17] запропонували перший набір метрик, відомий як набір метрик CK, для захоплення структурних аспектів об'єктно-орієнтованого програмування, який містить вісім метрик коду. Однак цей набір метрик містить дуже тривіальні метрики коду, що залежать від підрахунку характеристик. Абреу

та ін. [18, 22] запропонували вдосконалений набір метрик, відомий як набір метрик для об'єктно-орієнтованого проєктування (MOOD), який містить шість метрик коду для охоплення структурних аспектів об'єктно-орієнтованого проєктування. Лоренц та ін. [23] запропонували вдосконалений набір метрик у порівнянні з набором метрик MOOD, щоб краще відобразити структурні аспекти об'єктно-орієнтованого проєктування, який містить дванадцять метрик коду. Тегарден та ін. [24] запропонували три нові метрики коду — CLD, NOA та NOD — для захоплення зв'язку по успадкуванню в об'єктно-орієнтованому проєктуванні. Лі та ін. [25] запропонували чотири нові метрики коду на основі графа потоку інформації для захоплення згуртованості та зв'язності. Біман і Канг [26] запровадили дві нові метрики коду на рівні класу, а саме TCC і LCC, для вимірювання сили взаємозв'язку між усіма методами класу. Хендерсон-Селлер [27] запропонував три нові метрики коду — AID, LCOM1 та LCOM5 — для кращого захоплення зв'язку по успадкуванню та згуртованості класів, ніж метрики CK. Лі [28] запропонував шість нових метрик коду для захоплення структурних аспектів об'єктно-орієнтованого проєктування, а в 2002 році Бансия та Девіс [29] запропонували новий набір метрик для охоплення всіх структурних аспектів об'єктно-орієнтованого проєктування, таких як складність класів, зв'язність класів через успадкування та виклики методів, відсутність згуртованості між методами класу, абстракція, приховування та поліморфізм.

1.3 Постановка задачі

Актуальність дослідження полягає в постійно зростаючій складності програмного забезпечення, яке розробляється в умовах швидко змінюваного технологічного середовища. Сучасні програми часто мають велику кількість функціональних компонентів, інтеграцій і залежностей, що ускладнює їх тестування та виявлення дефектів. У таких умовах традиційні підходи до

управління якістю стають недостатніми, і виникає потреба в нових методах, які здатні адекватно реагувати на виклики сучасного програмування.

Одним із найбільш актуальних аспектів управління якістю програмного забезпечення є прогнозування дефектів, яке дозволяє передбачити й ідентифікувати потенційні проблеми до того, як вони вплинуть на кінцевого користувача. Це особливо важливо в умовах жорстких термінів виконання проектів, де затримки у виявленні дефектів можуть призвести до значних фінансових втрат і погіршення репутації компанії. Використання сучасних методів прогнозування дефектів може суттєво знизити ризики та підвищити ефективність розробки програмного забезпечення.

Дослідження, присвячені прогнозуванню дефектів, зазвичай зосереджені на аналізі існуючих моделей і алгоритмів, проте не завжди враховують специфіку об'єктно-орієнтованого програмування. У цьому контексті важливо розробити нові структурні метрики, які б адекватно відображали всі аспекти якості коду. Включення таких метрик до процесу прогнозування може підвищити точність моделей і, відповідно, зменшити кількість виявлених дефектів у програмному забезпеченні.

Сьогодні існує безліч підходів до прогнозування дефектів, але багато з них є тривіальними і недостатньо ефективними. Традиційні метрики, що базуються на підрахунках характеристик коду, часто не враховують складних аспектів об'єктно-орієнтованого програмування, таких як успадкування, поліморфізм і зчеплення між класами. Це обмежує їх застосування у реальних проектах, що вимагає пошуку нових рішень та підходів.

Використання методів машинного навчання для прогнозування дефектів у програмному забезпеченні є ще одним важливим аспектом, який підвищує актуальність даного дослідження. Алгоритми машинного навчання мають потенціал для обробки великих обсягів даних і можуть бути налаштовані на адаптацію до специфіки різних проектів. Це дозволяє не лише підвищити точність

прогнозів, а й зменшити час на їх отримання, що є критично важливим у умовах швидкого розвитку програмного забезпечення.

Серед основних викликів, з якими стикаються команди розробників, є необхідність постійного вдосконалення процесів тестування та управління проектами. Інтеграція нових методів прогнозування дефектів у ці процеси може допомогти підвищити ефективність роботи команд і знизити витрати на виправлення помилок. Це, у свою чергу, створює потребу в розробці нових моделей, що ґрунтуються на сучасних підходах, таких як CPSDP, з акцентом на якість коду та структурні метрики.

Враховуючи все вище зазначене, можна стверджувати, що розробка моделі прогнозування дефектів на основі машинного навчання є надзвичайно актуальною. Це дослідження не лише надає нові підходи до прогнозування дефектів, але й формує основу для подальших досліджень у цій галузі. Актуальність теми підкреслюється також потребою в адаптації розроблених методик до різних контекстів і специфікацій програмного забезпечення.

У підсумку, актуальність цього дослідження визначається складністю сучасного програмного забезпечення, необхідністю покращення прогнозування дефектів та впровадженням нових метрик і методів. Розроблені підходи можуть стати важливим інструментом для команд, що працюють у галузі програмного забезпечення, допомагаючи їм ефективно справлятися з викликами, що постають у процесі розробки та тестування програмних продуктів.

Отже, метою даної роботи є розробка моделі прогнозування дефектів програмних проєктів на основі алгоритмів машинного навчання, яка дозволить покращити точність і ефективність виявлення потенційних дефектів у програмному забезпеченні на етапі його розробки та підтримки.

Задачі, які необхідно вирішити для досягнення поставленої мети, включають:

1. Проаналізувати виклики управління складністю програмного забезпечення та сучасні інструменти прогнозування дефектів.

2. Провести огляд існуючих рішень для прогнозування дефектів у програмних проєктах.

3. Розробити модель прогнозування дефектів між проєктами на основі машинного навчання.

4. Дослідити різні класифікатори на основі машинного навчання для прогнозування дефектів.

5. Запропонувати нові структурні метрики для покращення об'єктно-орієнтованих систем.

6. Провести порівняльний аналіз продуктивності алгоритмів машинного навчання на основі метрик коду.

7. Проаналізувати чутливість та результати запропонованої моделі в контексті існуючих досліджень.

Висновки до розділу 1

1. Огляд літератури з прогнозування дефектів програмного забезпечення показує, що існуючі підходи переважно використовують структурні метрики коду, які охоплюють основні об'єктно-орієнтовані аспекти. Багато досліджень зосереджуються на оптимізації моделей машинного навчання (ML) та на пошуку відповідних метрик для покращення точності прогнозування дефектів. Одним із таких підходів є Cross-Project Software Defect Prediction (CPSDP), який використовує дані інших проєктів для тренування моделей, що робить його більш гнучким у реальних умовах.

2. Більшість існуючих рішень базується на наборах метрик, які часто є недостатньо точними для глибокого аналізу програмного забезпечення. Зокрема, у багатьох випадках використовуються тривіальні метрики на основі підрахунків характеристик коду, що не завжди відображає складні аспекти об'єктно-орієнтованого програмування. Проте в літературі з'являються нові пропозиції, що акцентують увагу на метриках, які більш ефективно відображають ключові

структурні властивості програмного забезпечення, такі як поліморфізм, абстракція та згуртованість класів.

3. Останні дослідження у сфері CPSDP включають нові алгоритми та метрики, що дозволяють значно покращити продуктивність моделей прогнозування дефектів. Особливо перспективними є рішення, які інтегрують семантичні характеристики коду, такі як CNN і TCNN, які досягають вищих результатів у порівнянні з традиційними підходами. Використання нових метрик, таких як ETCC, ECAC та ELCOCC, дозволяє суттєво підвищити якість моделей CPSDP, що робить їх кращими за існуючі аналоги на основі традиційних структурних метрик.

2 МОДЕЛЬ ПРОГНОЗУВАННЯ ДЕФЕКТІВ МІЖ ПРОЄКТАМИ

2.1 Модель прогнозування дефектів між проєктами

Далі буде представлено розроблена модель прогнозування дефектів між проєктами (CPSDP) на основі машинного навчання (ML). Метрики якості коду, які використовуються як вхідні характеристики, необхідні для кожної моделі SDP на основі ML, і ми пропонуємо новий набір метрик, щоб охопити всі структурні аспекти об'єктно-орієнтованої структури для розробки запропонованої моделі CPSDP. Для тестування нашої моделі обрано вісімнадцять проєктів з відкритим вихідним кодом, написаних мовою Java, вихідний код і інформація про дефекти яких доступні в репозиторії Promise [37].

На першому етапі видобуваються запропоновані метрики коду для кожного класу разом з іменами класів усіх обраних проєктів. На основі отриманих структурних метрик коду створюються нові набори даних без міток для всіх обраних проєктів, але для навчання моделі ML нам потрібні дані з мітками.

Тому витягнувши мітки класів із наборів даних репозиторію Promise для обраних проєктів і додаємо їх до новостворених наборів даних. На другому етапі нормалізуємо всі набори даних обраних проєктів за допомогою нормалізації Мін-Макс для встановлення масштабу. На третьому етапі для тестування моделі обирається один набір даних проєкту з усіх обраних наборів.

Для решти наборів даних проєктів виконується t-тест із тестовим набором даних, і на основі середнього значення t-статистики для запропонованих метрик коду обираються топ k наборів даних для підготовки навчальних даних.

У нашому випадку значення k становить 10. Усі записи з обраних k наборів даних об'єднуються для створення єдиного набору даних для навчання моделі ML. Нарешті, модель ML тренується на основі підготовленого навчального набору даних, і продуктивність тестується на основі результатів прогнозування добре навченої моделі ML на тестовому наборі даних.

Гіперпараметри налаштовуються на основі 5-кратної перехресної перевірки (5-fold CV) навчального набору даних, щоб уникнути перенавчання моделі, і для вибору найбільш відповідного алгоритму ML для запропонованої моделі CPSDP порівнюються вісім алгоритмів ML, таких як дерево рішень (DT), логістична регресія (LR), випадковий ліс (RF), адаптивне посилення (AdaBoost), екстремальне градієнтне посилення (XGBoost), штучна нейронна мережа (ANN), машина опорних векторів (SVM) та k-ближчих сусідів (KNN).

Запропонована модель CPSDP тестується на всіх обраних наборах даних проєктів, обираючи кожен набір як тестовий, та порівнюється з існуючими моделями на основі таких показників, як AUC, точність, precision, recall, f1-score та показник помилкових пропусків. Рисунок 2.1 демонструє загальний робочий процес нашої запропонованої моделі CPSDP.

Оскільки пропоновані нові структурні метрики коду для розробки покращеної моделі CPSDP, наступний розділ містить детальний опис запропонованих метрик коду.

Клас C = {V, F, IP} є базовим будівельним блоком будь-якого програмного забезпечення, написаного з використанням об'єктно-орієнтованого програмування. Пропонуємо метрики коду на рівні класу, щоб охопити структурні аспекти об'єктно-орієнтованого програмування.

Клас C = {V, F, IP} складається з трьох множин: V, F і IP. F — це множина методів/функцій F = {f0, f1,…, fn}, V — це множина полів/змінних V = {v0, v1,…, vm}, а IP — це множина успадкованих властивостей IP = {ip0, ip1,…, ipl}. Тут n, m і l — це кількість елементів у множинах V, F і IP відповідно.

Спочатку нам потрібно призначити ваги всім змінним і методам класів, щоб розрахувати значення запропонованих метрик коду для охоплення структурних аспектів об'єктно-орієнтованого програмування. Ваги полів класу призначаються на основі рівняння (2.1).

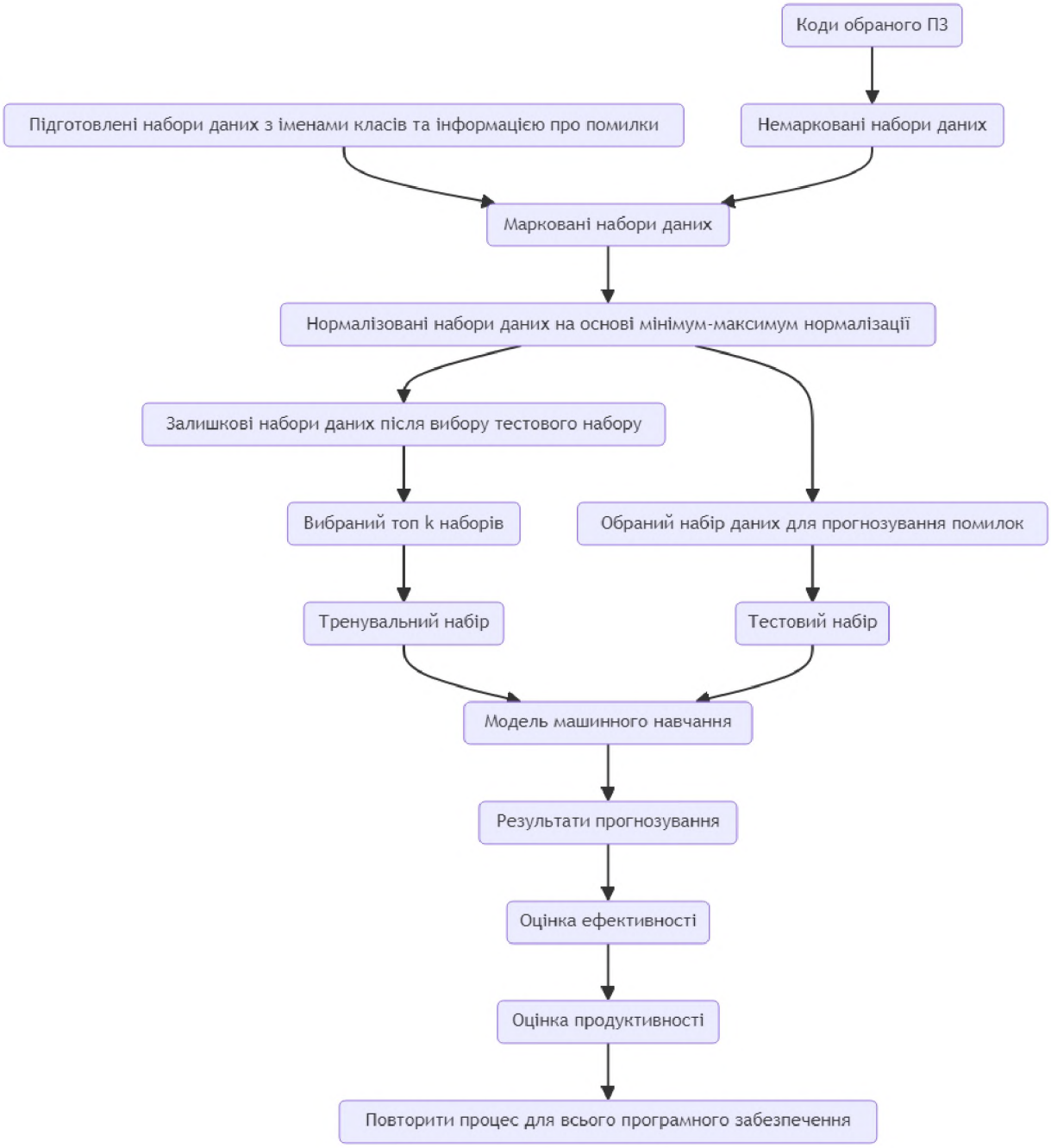

Рисунок 2.1 - Загальна структура моделі CPSDP

$$W(v_i) = \begin{cases} 1, void \\ a, Strings \\ b, numbers(integers\ and\ fractions) \\ c, Data\ structures(collections) and\ arrays \\ \sum_{v \quad \in V^{OC}} w(v), \quad objects\ of\ other\ classes \\ 2, others \end{cases} where\ c > b > a > 2 \tag{2.1}$$

де $W_{(v_i)}$ - вага i-тої змінної v_i класу;

$\sum_{v \in V^{OC}} w(v)$ - вага об’єкта іншого класу, визначеного в поточному класі;

V^{OC} - множина змінних іншого класу.

a - вага рядків і символів;

b — вага цілих і дробових змінних;

a і c — ваги об'єктів складних структур даних, таких як колекції, дерева та масиви.

Ваги змінних типу *void* дорівнюють 1, а для інших типів змінних, таких як URL-адреси та шляхи, ваги дорівнюють 2. Ваги змінних повинні відповідати умові $c > b > a > 2$.

Оскільки a представляє ваги рядків і символів, і всі операції з рядками в об'єктно-орієнтованих мовах програмування обробляються вбудованими функціями, з іншого боку, b представляє ваги цілих чисел і чисел з плаваючою точкою, які використовуються в арифметичних виразах та умовних операторах, а арифметичні вирази мають більші шанси на появу помилок. Тому ваги цілих чисел і чисел з плаваючою точкою повинні бути більшими за ваги рядків і символів. Крім того, ваги складних структур даних, таких як масиви, зв'язані списки, дерева, стеки та черги, представлені змінною c у рівнянні (2.1), повинні бути більшими за ваги цілих чисел і дробів, оскільки операції з комплексними структурами даних також складні, такі як множення матриць, сортування масивів, векторний добуток тощо. Таким чином, ваги змінних у рівнянні (2.1) повинні відповідати умові $c > b > a > 2$.

Далі пропонується використати метрику коду на рівні методу, яка називається "зважена складність шляхів" (WPC), розширену версію цикломатичної складності. Існують деякі недоліки цикломатичної складності, такі як те, що вона не враховує інформацію про рівень вкладеності інструкцій, не розрізняє оператори циклів від умовних операторів і лише знаходить загальну кількість лінійно незалежних шляхів у функціях. Щоб усунути ці обмеження цикломатичної складності, пропонована нова метрика коду на рівні функцій/методів, яка

називається "зважена складність шляхів" (WPC). Вона призначає ваги всім інструкціям методу на основі рівня їх вкладеності та спеціальні ваги умовним і циклічним операторам у графі потоку керування, і знаходить суму ваг для всіх шляхів.

Припустимо, що в графі потоку керування будь-якого методу/функції f є m кількість шляхів, а в і-му шляху є n_i інструкцій. Множина шляхів у графі потоку керування будь-якого методу f представлена як $P = \{p_0, p_1, \ldots, pm\}$, інструкції будь-якого шляху p_i представлені як $I_{pi} = \{ip_{i0}, ip_{i1}, \ldots, ip_{ini}\}$, рівень вкладеності кожної інструкції шляху p_i представлений як $NL_{pi} = \{nl_{pi0}, nl_{pi1}, \ldots, nl_{pini}\}$.

Рівняння (2.2) можна використати для розрахунку значення WPC будь-якого методу.

$$WPC(f) = \sum_{p \in P} \sum_{i \in I^p} w(i) * nl(i) \quad , \tag{2.2}$$

де $p \in P$ представляє будь-який шлях p з множини шляхів P у графі потоку керування будь-якого методу f, і представляє інструкцію будь-якого шляху p.

$w(i)$ - вага інструкції;

$nl(i)$ - рівень вкладеності інструкції.

Усі інструкції мають однакову вагу w, за винятком циклічних і умовних операторів.

wc - ваги умовних операторів;

wl - ваги циклічних операторів.

Оператори switch case та тернарні оператори вважаються умовними операторами. Ваги циклів, умов та інших операторів повинні задовольняти умову $w < wc < wl$. Ваги умовних операторів більше, ніж у інших операторів, оскільки умовні оператори, такі як if-else, switch-case та тернарні оператори, використовуються для створення розгалужень. Ваги циклічних операторів, таких

як for, while та do-while, більші за ваги умовних операторів, оскільки вони обробляють більш складну логіку, ніж умовні оператори.

WPC має три переваги перед цикломатичною складністю: по-перше, вона враховує довжину шляхів у графі потоку керування, і різні шляхи роблять різний внесок залежно від кількості інструкцій у них, тоді як цикломатична складність лише визначає загальну кількість шляхів у графі. По-друге, вона враховує рівень вкладеності кожної інструкції в шляхах графа потоку керування, що не відображається в цикломатичній складності. По-третє, вона розрізняє умовні, циклічні та звичайні інструкції, тоді як цикломатична складність цього не робить.

Далі використовується чотири метрики коду на рівні класу для відображення структурних аспектів об'єктно-орієнтованої системи, використовуючи рівняння (2.1) та (2.2), які використовуються для розробки нашої запропонованої моделі CPSDP. Виведення запропонованих метрик коду наведені нижче.

2.1.1 Розширена загальна складність класу та зчеплення між класами

Клас C може бути представлений як множина полів, методів і успадкованих властивостей. Усі ці три компоненти відіграють важливу роль у визначенні складності класу C в об'єктно-орієнтованій системі. Багато структурних метрик коду було запропоновано в останні роки для визначення складності класу в об'єктно-орієнтованій структурі. Однак вони є недостатніми для визначення справжньої складності класу, оскільки ці тривіальні метрики не відображають детальної інформації. Щоб подолати недоліки існуючих метрик, запропоновано нову метрику коду під назвою "розширена загальна складність класу" (ETCC), яка є покращеною версією метрики TCC, для визначення складності класу в об'єктно-орієнтованій системі. Призначається ваги всім публічним і приватним полям і методам класу на основі рівнянь (2.1) та (2.2), а потім обчислюємо їх суму, щоб знайти фактичне значення ETCC для класу C, як показано в рівнянні:

$$ETCC(C)= \quad \sum_{i=1}^{n} w(v_i) + \quad \sum_{j=1}^{m} WPC(f_j) + \quad \sum_{k=1}^{l} ETCC(o_k) \quad , \quad (2.3)$$

де *WPC(fj)* - зважена складність шляхів для *j*-го методу/функції класу C. Зважену складність шляхів методу (WPC) можна розрахувати за допомогою рівняння (2.2).

$w(v_i)$ - вагу *i*-го поля класу C.

Поле може бути публічним або приватним, і вага призначається на основі типу даних поля, як показано в рівнянні (2.1). Якщо поле є об'єктом іншого класу, тоді вага поля є сумою всіх публічних і приватних полів цього класу.

$ETCC(o_k)$ - розширена загальну складність класу для успадкованих класів і реалізованих інтерфейсів.

Існує багато метрик коду, таких як CBO і RFC, для вимірювання зчеплення між класами. Однак усі ці метрики є досить тривіальними й просто підраховують вхідні та вихідні виклики методів, що недостатньо для визначення реальної сили зчеплення. Тому в цьому дослідженні пропонується нова метрика коду під назвою "розширене зчеплення між класами" (ECAC), яка є розширенням CAC [43], і враховує параметри та їх типи даних, передані через виклики методів, їх змінні повернення, вагу об'єктів інших класів, оголошених у класі, та ваги публічних полів. Важливо враховувати всі ці фактори.

Припустимо, клас отримує параметри з вхідним викликом методу, або між класами (інтер-клас), або всередині класу (інтра-клас). У такому випадку значення приватних полів класу може бути змінене отриманими параметрами. Якщо клас отримує параметр повернення з вихідного виклику, то можливе призначення поверненого значення будь-якому приватному полю класу, і помилкове значення може призвести до дефекту. Якщо об'єкт класу C1 оголошено в класі C2, то між класами C1 і C2 існує пряме зчеплення. Якщо клас містить будь-яке публічне поле, то до нього можна безпосередньо звертатися через об'єкт класу в іншому класі без

виклику методу. Тому публічні поля також потрібно враховувати під час обчислення сили зчеплення будь-якого класу.

Існують два типи викликів методів, які пояснюються нижче. Якщо один клас пов’язаний з іншим через виклик методу, це називається міжкласовим зчепленням (інтер-клас), що показано вхідним або вихідним ребром на рисунку 2.2. З іншого боку, якщо клас пов'язаний сам із собою через виклик методу, це називається внутрішньокласовим зчепленням (інтра-клас), що показано самопетлею на рисунку 2.2.

У випадку міжкласового зчеплення, якщо виклик методу є вхідним, отримані параметри можуть змінити значення приватних полів класу, тому для розрахунку зчеплення враховуємо лише отримані параметри. З іншого боку, у випадку вихідного виклику методу лише значення повернення можуть змінити значення приватних полів класу, тому для обчислення зчеплення враховуємо лише значення повернення. Якщо один метод класу викликає інший метод того ж класу, це називається внутрішньокласовим зчепленням. У цьому сценарії як передані параметри, так і значення повернення можуть змінити приватні поля класу, тому обидва повинні враховуватися під час обчислення зчеплення цього класу.

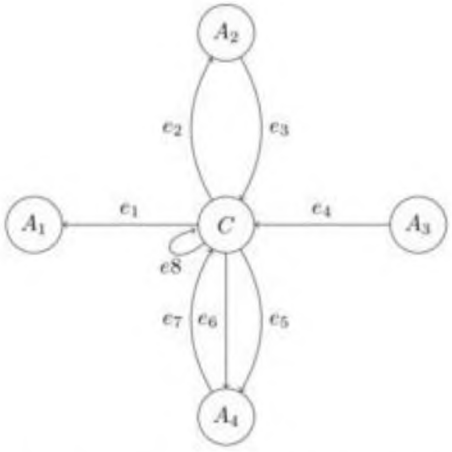

Рисунок 2.2 - Малий приклад зчеплення.

Припустимо, клас C має всього m ребер, з яких d є вихідними ребрами, а m − d є вхідними ребрами, як показано на рисункку 2.2. Самопетля представляє

внутрішньокласовий виклик методу (intra-class method call). Припустимо, клас C має загалом внутрішньокласових викликів методів, і кожен виклик методу має n параметрів, що передаються, та значення, яке повертається.

Припустимо, клас C також має r публічних полів і s об'єктів інших класів; тоді ECAC можна розрахувати на основі Рівняння

$$ECAC(C) = \sum_{i=1}^{d}\sum_{j=1}^{n} w(v_{i,j}) + \sum_{i=d+1}^{m} w(r_i) + \sum_{i=1}^{l}\sum_{j=1}^{n} w(v_{i,j}) + \sum_{i=1}^{l} w(r_i) + \sum_{i=1}^{r} w(pv_i) + \sum_{i=1}^{s} TCC(po_i) \quad , \qquad (2.4)$$

де $w(v_{i,j})$ - вага j-го параметра для i-го виклику методу;

$w(pv_i)$ - вага публічної змінної;

$TCC(po_i)$ - загальна вага об'єкта іншого класу, оголошеного в класі C;

$w(r_i)$ - вага змінної повернення.

2.1.2 Розширена відсутність згуртованості між пов'язаними компонентами

Існує багато метрик коду для обчислення згуртованості класів в об'єктно-орієнтованих системах, таких як LCOM, LCOM3, Coh тощо. Однак ці метрики враховують лише пари згуртованих методів і не обчислюють справжню силу згуртованості. Крім того, вони або не враховують приватні методи класу, або розглядають приватні методи окремо від публічних. Однак під час обчислення справжньої сили згуртованості приватні методи повинні бути поєднані з публічними, оскільки їх можна викликати лише з публічних методів класу. Щоб вирішити всі ці проблеми існуючих метрик згуртованості, запропоновано нову метрику коду під назвою ELCOCC.

Припустимо, клас C містить чотири змінні/поля $V = \{v_1, v_2, v_3, v_4\}$ з різними типами даних і п'ять методів $F = \{f_1, f_2, f_3, f_4, f_5\}$, як показано на Рисунку 2.3.

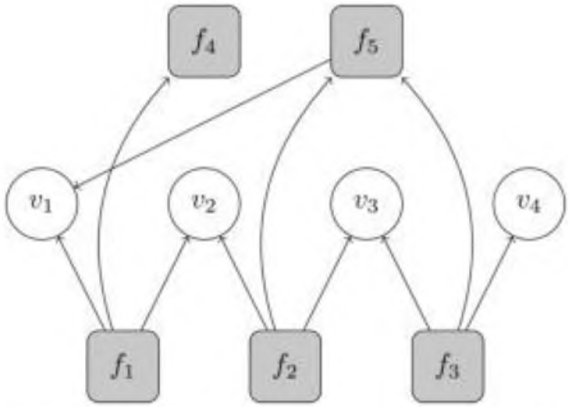

Рисунок 2.3 - Малий приклад згуртованості.

На рисунку 2.3 поля показані у вигляді кіл, а методи — у вигляді прямокутників із закругленими кутами. З усіх п'яти методів два методи, f_4 і f_5, є приватними, а три методи, f_1, f_2 та f_3, є публічними. Приватні методи викликаються лише з публічних методів. Якщо будь-який приватний метод викликається з публічного методу, його слід розглядати як один пов'язаний компонент.

На рисунку 2.3 є три пов'язані компоненти: Comp = {$comp_1$ = {f_1, f_2}, $comp_2$ = {f_1, f_2, f_3}, $comp_3$ = {f_1, f_3, f_4}}. Компонент $comp_1$ об'єднує f_1 і f_4 та отримує доступ до полів v_1 та v_2. Компонент $comp_2$ об'єднує публічний метод f_2 і приватний метод f_5 та отримує доступ до полів v_1, v_2 та v_3. Компонент $comp_3$ об'єднує публічний метод f_3 та f_5 та отримує доступ до полів v_1, v_3 та v_4. Ваги всім полям призначаються на основі рівняння (2.1).

Після ідентифікації всіх пов'язаних компонентів у класі та їхніх доступних приватних полів, ELCOCC можна розрахувати:

$$ELCOCC(C) = \frac{\sum_{i=1}^{l}\sum_{j=i+1}^{l}\sum_{f\in comp_i\cap comp_j} w(v)}{l^*\frac{l-1}{2}*\sum_{i=1}^{l} w(v_i)}, \quad (2.5)$$

де l - загальна кількість полів у класі;

$w(v)$ - вага поля;

$comp_i \cap comp_j$ - спільні поля двох пов'язаних компонентів $comp_i$ та $comp_j$.

2.1.3 Розширене співвідношення складності перевантажених та успадкованих методів

Поліморфізм є однією з найважливіших особливостей об'єктно-орієнтованого програмування. Перевантаження операторів і функцій — це два важливі концепти в цій категорії, які можуть відігравати важливу роль у прогнозуванні дефектів програмного забезпечення (SDP). Дуже мало метрик коду було запропоновано на основі цієї концепції, і всі вони захоплюють тривіальну інформацію, таку як кількість перевантажених методів (NOM). Тому існує необхідність у кращому відображенні поліморфізму, а не в тривіальному підрахунку перевантажених методів. Це дослідження пропонує нову структурну метрику коду, що отримала назву "розширене співвідношення складності перевантажених методів" (EOMCR), яка є розширеною версією OMCR для кращого відображення поліморфних характеристик об'єктно-орієнтованого програмування. Вона відображає співвідношення складності перевантажених методів до загальної складності класу.

Припустимо, клас C складається з кількох перевантажених методів, представлених множиною $O = \{o_1, o_2, \ldots, o_m\}$, і кожен перевантажений метод має l_i копій, таких як $O_i = \{f_1, f_2, \ldots, f_{li}\}$. Загалом у класі є n методів, представлених множиною $F = \{f_1, f_2, \ldots, f_n\}$, тоді EOMCR можна розрахувати за допомогою рівняння (2.6):

$$EOMCR(C) = \frac{\sum_{i=1}^{m} \sum_{f \in O_i} WPC(f)}{\sum_{i=1}^{n} WPC(f_i)}, \quad (2.6)$$

де *WPC(f)* - зважена складність шляхів для функції/методу, яку можна розрахувати за допомогою рівняння (2.2);

O_i - множина перевантажених методів *i*-го типу (всі методи множини O_i мають однакову назву, але різну кількість або тип аргументів);

m - загальна кількість множин перевантажених методів;

n - представляє загальну кількість методів класу.

Перевантажені методи можуть включати перевантаження методів або операторів, але в Java є лише перевантаження методів.

Успадкування також є однією з найважливіших особливостей об'єктно-орієнтованого програмування, яке відіграє важливу роль у прогнозуванні дефектів програмного забезпечення (SDP). У недавньому минулому було запропоновано багато структурних метрик коду для відображення зчеплення за успадкуванням в об'єктно-орієнтованій структурі програмного забезпечення. Однак ці метрики коду є тривіальними та не охоплюють глибокої інформації. Тому існує потреба в кращому захопленні зчеплення за успадкуванням. Це дослідження пропонує нову метрику під назвою "розширене співвідношення складності успадкованих методів" (EIMCR), яка є покращеною версією нашої раніше запропонованої метрики успадкування IMCR, щоб краще відображати зчеплення за успадкуванням порівняно з існуючими тривіальними метриками.

Якщо клас C успадковує властивості класу C', то клас C містить три типи методів: перевизначені, успадковані та новододані методи. Новододані та перевизначені методи містять нову логіку, яка є оригінальною реалізацією класу C, а успадковані методи містять стару реалізацію, вже виконану в базовому класі C'. Будь-яка помилка через неструктурований та складний код у новододаних і перевизначених методах залежить від складності класу C. З іншого боку, будь-яка помилка в успадкованих методах залежить від базового класу C', оскільки фактичний код успадкованих методів міститься в базовому класі C'. Тут відображено співвідношення складності успадкованих методів до всіх методів класу C, оскільки наша мета — захопити кількість коду, успадкованого від інших класів.

Припустимо, клас C містить множину успадкованих методів $I = \{f_1, f_2, \ldots, f_n\}$, множину перевизначених методів $O = \{f_1, f_2, \ldots, f_m\}$ та множину новододаних

методів N = {f_1, f_2,…, f_l}, тоді EIMCR можна розрахувати за допомогою рівняння (2.7).

$$EIMCR(C) = \frac{\sum_{f \in I} WPC(f)}{\sum_{f \in N} WPC(f) + \sum_{f \in O} WPC(f) + \sum_{f \in I} WPC(f)}, \quad (2.7)$$

де *WPC(f)* - зважена складність шляхів методу/функції *f*.

2.1.4 Категорія класу

Кожен проєкт, розроблений в IT-індустрії, має певну специфічну область. Наприклад, проєкти, такі як Putty та Wire Shark, спеціально призначені для роботи з мережами й складаються з багатьох класів, які виконують завдання, пов'язані з мережевими підключеннями, і, відповідно, включають сокетне програмування. З іншого боку, класи, пов'язані з графічними інтерфейсами (GUI), містять багато методів обробки подій, таких як події клавіатури та миші. Отже, необхідно знаходити подібні типи проєктів/класів для кращого прогнозування дефектів, і тому була розроблена нова метрика коду під назвою "категорія класу" (COC), яка служить цій меті. Однак це є суб'єктивним питанням і може становити загрозу для достовірності.

Метрика коду категорії класу (COC) розроблена для того, щоб представляти категорію класу на основі типу логіки, яка ділить класи на вісім категорій, і кожній категорії присвоюється унікальне число. Один клас може належати до кількох категорій; у цьому випадку значення COC дорівнює сумі унікальних чисел усіх категорій.

Рівняння (2.8) представляє унікальні числа всіх категорій, використаних у цій статті.

У Рівнянні (2.8) всі вісім категорій представлені одним цілим числом. Наша перша категорія містить усі абстрактні класи та інтерфейси, оскільки ці класи не

містять логіки. Друга категорія містить усі класи, які виконують введення/виведення (I/O). Введення/виведення може здійснюватися на основі файлів і пристроїв I/O, таких як монітор і клавіатура. Третя категорія містить усі класи, орієнтовані на дані. Ці типи класів містять велику кількість полів та їхні методи get/set. Проте жодне завдання не виконується над цими змінними, окрім встановлення та отримання їхніх значень. Четверта категорія містить класи, які взаємодіють з мережевими пристроями за допомогою сокетного програмування. П'ята категорія містить усі класи, які реалізують багатопоточну логіку. У Java всі класи, які або розширюють клас Thread, або реалізують інтерфейс Runnable, підпадають під цю категорію. Шоста категорія містить усі класи, які містять логіку графічного інтерфейсу користувача (GUI) та обробку подій. Сьома та восьма категорії класів засновані на складності логіки класу. Якщо будь-який метод класу має рівень вкладеності більше двох і більше п'яти циклів або умовних операторів if-else, то вважаємо це складною логікою. В іншому випадку клас вважається простим з точки зору логіки. Якщо будь-який клас не належить до жодної з цих категорій, присвоюємо йому вагу 0, що означає інші категорії. Варто зазначити, що ці ваги не є фактичними вагами класу, а представляють категорію класу, і один клас може належати до кількох категорій.

$$CAT = \begin{cases} 0, others \\ 1, Interface\ or\ abstract\ class \\ 2, \quad I/O \\ 4, get\ and\ set\ functions\ only \\ 8, Networking \\ 16, Multi-threading \\ 32, GUI\ logic \\ 64, Simple\ logic \\ 128, complex\ logic \end{cases}, \tag{2.8}$$

Припустимо, клас C містить як GUI-логіку, так і логіку введення/виведення. У такому випадку значення COC для цього класу буде дорівнювати сумі чисел обох категорій, наприклад, у випадку GUI та введення/виведення значення COC для класу C становитиме 34 (32+2 = 34).

Припустимо, що клас C належить до k категорій із доступних n категорій, представлених вектором B з нулями та одиницями. Якщо значення i-го елемента вектора B дорівнює одиниці, то цей клас належить до i-ї категорії і навпаки. На основі цього припущення, COC класу C можна розрахувати за допомогою :

$$COC(C) = \sum_{i=1}^{n} \overrightarrow{B}_i * CAT_i, \tag{2.9}$$

де CAT_i - *i*-та категорію з рівняння (2.8);

B — вектор, що складається з нулів та одиниць, які вказують, чи належить клас C до *i*-ї категорії чи ні. Якщо значення COC(C) дорівнює нулю, тоді клас C належить до інших категорій.

2.2 Класифікатори на основі машинного навчання

Наведені нижче алгоритми машинного навчання використовуються для експериментів з метою порівняння ефективності запропонованої моделі CPSDP з існуючими дослідженнями. Застосовуються чотири локальні класифікатори: DT (дерево рішень), LR (логістична регресія), SVM (метод опорних векторів) та KNN (k найближчих сусідів); три ансамблеві класифікатори: RF (випадковий ліс), AdaBoost (адаптивне посилення) і XGBoost (екстремальне градієнтне посилення); а також одношарова нейронна мережа. Гіперпараметри цих алгоритмів машинного навчання налаштовуються за допомогою 5-кратної перехресної перевірки (5-fold CV). Нижче подано пояснення обраних алгоритмів та їх гіперпараметрів.:

2.2.1 Дерево рішень

Дерево рішень (DT) — це алгоритм керованого машинного навчання, який підтримує ієрархічну структуру, рекурсивно поділяючи набір даних на підмножини до тих пір, поки не буде виконана умова зупинки. Перший параметр налаштування — критерій поділу, перевіряється як індекс Джині, так і інформаційну вигоду. Другий параметр — максимальна глибина, і перевірялись на всі значення від 2 до 32.

2.2.2 Логістична регресія

Логістична регресія (LR) — це найпростіший алгоритм машинного навчання, заснований на логарифмічних відношеннях. Має лише один параметр налаштування, який називається регуляризацією, і перевіряються всі три варіанти: L1-норма (гребенева регресія), L2-норма (LASSO-регресія) і їх комбінація (еластична мережа).

2.2.3 Метод опорних векторів

Метод опорних векторів (SVM) — це складний алгоритм машинного навчання, який створює межі для розділення точок різних класів. З доступних чотирьох ядер (лінійне, сигмоїдне, RBF і поліноміальне) використовуємо ядро RBF, оскільки воно перевершує всі інші ядра для нелінійно роздільних класів. Перевірені значення гіперпараметра гамма: 10, 1, 0.1, 0.01. Значення гіперпараметра C фіксоване на рівні 1.

2.2.4 K найближчих сусідів

K найближчих сусідів (KNN) — це непараметричний алгоритм машинного навчання, який присвоює мітки тестовим зразкам на основі k найближчих зразків, вибраних із навчального набору даних за допомогою евклідової відстані. Стратегія більшості голосів присвоює мітку тестовому зразку на основі вибраних

найближчих сусідів. Кількість найближчих сусідів є гіперпараметром цього алгоритму, перевірили всі непарні значення в діапазоні від 3 до 21.

2.2.5 Штучна нейронна мережа

Штучна нейронна мережа (ANN) натхненна роботою людського мозку. Це мережа штучних нейронів, організована в багатошарову структуру, що складається з вхідного шару, одного або кількох прихованих шарів і одного вихідного шару. Кожен нейрон має вагу, яку потрібно оптимізувати для побудови гарної моделі. Кількість прихованих шарів, кількість нейронів у кожному прихованому шарі та активаційна функція нейронів є гіперпараметрами нейронної мережі. Використовуємо одношарову нейронну мережу, тому прихований шар буде один. Перевірені всі значення від 5 до 50 (з кроком 5) для розміру прихованого шару. Перевірено чотири активаційні функції: лінійна (identity), сигмоїдна (logistic), тангенс (tanh), і relu.

2.2.6 Випадковий ліс

Випадковий ліс (RF) — це ансамблевий метод, що є комбінацією кількох дерев рішень. Він використовує метод бутстрепа та випадкове підпросторове вибіркове об'єднання для генерації кількох слабких компонентів (дерев рішень). Кожен компонент дає своє незалежне передбачення, а кінцева мітка призначається на основі стратегії більшості голосів. Перевірені всі непарні значення розміру ансамблю в діапазоні від 11 до 31, а значення максимальних ознак встановлено на d/3, де d — це розмірність набору даних.

2.2.7 Адаптивне посилення

AdaBoost, також відомий як адаптивне посилення, є послідовним ансамблевим методом, розробленим на основі концепції бустингу. Будь-який локальний класифікатор може бути використаний як компонент для навчання

моделі AdaBoost, але тут використовуємо рішення-штамп як наш компонент. Перевірено всі непарні значення кількості оцінювачів у діапазоні від 11 до 21.

2.2.8 Екстремальне градієнтне посилення

Це також послідовний ансамблевий метод, розроблений на основі бустингу, як і AdaBoost. Це покращений варіант градієнтного бустингу, де наступний компонент навчається на основі залишкової помилки попереднього. На відміну від AdaBoost, у XGBoost використовується лише дерево рішень як компонент навчання. Він має три основні гіперпараметри: розмір ансамблю, максимальну глибину дерева та швидкість навчання. Перевірено всі непарні значення розміру ансамблю в діапазоні від 3 до 21, усі значення максимальної глибини дерева від 3 до 15, а стандартне значення швидкості навчання становить 0.3.

2.2.9 Згорткова нейронна мережа

Це варіант стандартної нейронної мережі з додаванням згорткового шару. CNN досягла великих успіхів у галузі прогнозування дефектів програмного забезпечення (SDP). У цьому дослідженні розмір партії встановлено на 32, кількість ітерацій — на 15, розмір вбудовування — 30, кількість прихованих нейронів — 100 і 10 фільтрів, кожен з довжиною 5 [44].

2.2.10 Передавана згорткова нейронна мережа

Передавана згорткова нейронна мережа - це покращена версія стандартної CNN, де до стандартної CNN додано шар відповідності. Усі параметри TCNN такі ж, як і для CNN, як зазначено в [42].

2.3 Запропоновані структурні метрики для покращення об'єктно-орієнтованих систем

Далі представлені деталі структурних метрик коду, використаних у наборах даних Promise, а також запропонований набір метрик для вибраних проєктів. Репозиторій Promise містить код більше десяти проєктів, написаних мовою об'єктно-орієнтованого програмування (Java), та їхні набори даних з інформацією про дефектні класи. Ці набори даних створені шляхом вибору двадцяти структурних метрик коду з літератури, що охоплюють усі структурні аспекти об'єктно-орієнтованого програмування.

Таблиця Ж.1 (див. додаток Ж) представляє деталі цих структурних метрик коду, такі як назва, тип та опис. Оскільки це дослідження пропонує набір метрик для кращого відображення структурних аспектів об'єктно-орієнтованого програмного забезпечення, таблиця 3.1 представляє деталі запропонованих метрик коду, такі як назва, тип і опис.

Складність класу, згуртованість методів класів, зчеплення між класами через виклики методів, успадкування та поліморфізм є п’ятьма основними аспектами проєктів, розроблених з використанням об'єктно-орієнтованих мов програмування, таких як Java. Запропонований набір метрик охоплює всі п’ять аспектів об'єктно-орієнтованої системи. ETCC розраховує складність класу, ECAC обчислює зчеплення між класами через виклики методів, ELCOCC вимірює згуртованість методів класу, EIMCR оцінює успадковану інформацію класу, а EOMCR оцінює поліморфізм, такий як перевантаження методів і операторів (хоча перевантаження операторів у Java не існує, запропонована метрика це охоплює).

Більше того, запропоновані метрики коду використовують WPC як основу для визначення складності методів, а WPC є розширеною версією цикломатичної складності, яка враховує рівень вкладеності, довжину лінійно незалежних шляхів, кількість лінійно незалежних шляхів і диференціює звичайні, розгалужені та циклічні оператори. Отже, запропонований набір метрик, показаний у таблиці 2.1, охоплює всі аспекти об'єктно-орієнтованої структури кращим чином у порівнянні з тривіальними метриками, що базуються на підрахунку, наведеними в таблиці Ж.1. Таким чином, очікується, що розробка моделі CPSDP на основі запропонованого

набору метрик перевершить існуючі моделі, оскільки продуктивність алгоритмів машинного навчання значною мірою залежить від якості ознак.

Запропоновано шість структурних метрик коду для кращого відображення об'єктно-орієнтованих властивостей проєктів у порівнянні з існуючими метриками коду, що вимагає перевірки ефективності запропонованих метрик. У цьому розділі порівнюються запропоновані метрики з існуючими метриками на основі досягнутого значення AUC за допомогою методу кривої ROC [40, 45] та VARL (значення допустимого рівня ризику) [46], після чого проводиться ранжування. Порівняння запропонованих структурних метрик проводиться за категоріями з існуючими метриками, наприклад, ETCC призначена для оцінки складності класу й порівнюється з існуючими метриками складності, такими як WMC, NPM, LOC, AMC, MAX_CC та AVG_CC.

Таблиця 2.1 - Запропонований набір метрик коду

Метрика коду	Тип метрики коду	Назва та опис
ETCC	Складність	Називається розширеною загальною складністю класу, покращеною версією TCC. Призначена для визначення складності класу на основі публічних, приватних та успадкованих полів і методів.
ECAC	Зчеплення	Називається розширеним зчепленням між класами, покращеною версією зчеплення між класами. Вона враховує зважені параметри внутрішніх і зовнішніх викликів, а також значення повернення.
ELCOCC	Згуртованість	Розширена версія LCOCC, призначена для визначення зв'язності між методами класу на основі спільних полів, з урахуванням типів даних полів.
EIMCR	Успадкування	Називається розширеним співвідношенням складності успадкованих методів, покращеною версією IMCR.
EOMCR	Поліморфізм	Називається розширеним співвідношенням складності перевантажених методів, розширеною версією OMCR.
COC	Категорія логіки	Називається категорією класу. Вона призначена для категоризації класу на основі реалізованої логіки.

Таблиця Ж.2 (див. додаток Ж) надає порівняння за категоріями на основі методів ROC та VARL. Запропонована метрика складності (ETCC) посідає перше місце за обома методами, як VARL, так і ROC, серед усіх метрик складності. Метрика зчеплення (ECAC) також посідає першу позицію за обома методами серед усіх обраних метрик зчеплення. Метрика згуртованості (ELCOCC) займає перше місце за методом VARL і друге за ROC, хоча різниця між LCOM і ELCOCC

незначна. Метрики успадкування (EIMCR) та поліморфізму (EOMCR) також займають перші місця за обома методами. Метрика COC посідає друге місце після EOMCR за методом VARL, але нижче за MOA і DAM за ROC. Обрані метрики, які займають перші місця у своїх категоріях, забезпечують покриття всіх аспектів об'єктно-орієнтованої системи для розробки якісної моделі CPSDP.

Висновки до розділу 2

1. У цьому розділі розроблено та представлено модель прогнозування дефектів між проєктами (CPSDP) на основі машинного навчання. Для ефективної роботи моделі було запропоновано новий набір структурних метрик коду, що дозволяє більш точно оцінити якість об'єктно-орієнтованого програмного забезпечення. Ці метрики охоплюють різні аспекти складності класів, зчеплення між класами, згуртованості методів, успадкування та поліморфізму, що робить їх більш придатними для прогнозування дефектів порівняно з існуючими тривіальними метриками.

2. Проведений аналіз показав, що модель CPSDP, яка використовує запропоновані метрики, дозволяє значно покращити процес прогнозування дефектів програмного забезпечення. Для тестування моделі було обрано 18 проєктів з відкритим вихідним кодом з репозиторію Promise, де нормалізація даних та налаштування гіперпараметрів виконувались за допомогою перехресної перевірки. Також були протестовані різні алгоритми машинного навчання, і модель XGBoost показала кращі результати за багатьма показниками ефективності.

3. Запропоновані структурні метрики коду та обраний алгоритм XGBoost забезпечили високу якість моделі прогнозування дефектів. У порівнянні з іншими підходами, запропоновані рішення продемонстрували суттєве покращення за такими показниками, як AUC, точність та f1-score. Модель CPSDP підтвердила свою ефективність і може бути рекомендована для впровадження в процеси автоматизованого тестування програмного забезпечення.

3 ПРАКТИЧНЕ ВИКОРИСТАННЯ ОТРИМАНИХ НАУКОВИХ РЕЗУЛЬТАТІВ

3.1 Порівняння продуктивності алгоритмів машинного навчання для прогнозування дефектів на основі метрик коду

Обрано вісімнадцять проєктів з відкритим вихідним кодом, написаних мовою програмування Java, вихідні коди яких, інформація про дефектні класи та існуючі структурні метрики доступні в репозиторії Promise для вибраних проєктів [37]. Ці проєкти були обрані для цього дослідження, оскільки інформація про дефекти (мітки класів), інформація про існуючі метрики для кожного класу та вихідні коди Java цих проєктів також доступні. Код Java потрібен для вилучення запропонованих метрик коду, а інформація про дефекти для кожного класу потрібна для маркування набору даних, створеного на основі запропонованих метрик коду. Інформація про існуючі метрики коду разом із мітками класів потрібна для порівняння запропонованої моделі CPSDP з існуючою моделлю CPSDP, заснованою на метриках.

Таблиця 3.1 містить інформацію про вибрані проєкти, такі як загальна кількість дефектних класів, загальна кількість недефектних класів, загальна кількість полів, загальна кількість методів, назва проєкту та версія тощо.

Обрано вісім алгоритмів машинного навчання для експериментів з метою відповісти на це дослідницьке запитання: DT, LR, SVM, KNN, RF, Adaboost, XGBoost, і ANN. Ці алгоритми машинного навчання тренуються на основі існуючих структурних метрик коду для збору результатів, а найвідповідніший алгоритм для CPSDP обирається на основі AUC і точності (accuracy). Зібрані значення AUC та точності для всіх проєктів представлені в таблицях Г.1 та Г.2 (див. додатог Г).

Таблиця Г.1 представляє зібрані значення AUC після застосування алгоритмів машинного навчання до вибраних проєктів. Таблиця показує найкращі результати жирним шрифтом. Жоден алгоритм не демонструє найкращі результати

для всіх вибраних проєктів. Таким чином, не можна зробити остаточний висновок щодо найкращого алгоритму машинного навчання на основі таблиці Г.1, але можна сказати, що, в середньому, випадковий ліс (RF) показує кращі результати, ніж інші алгоритми машинного навчання, оскільки середнє значення AUC, досягнуте випадковим лісом, становить 0.72265, що є вищим за всі інші алгоритми. У випадку 9 з 18 проєктів випадковий ліс дає найкращі результати, як показано жирним шрифтом у таблиці Г.1.

Таблиця 3.1 - Вибрані проєкти для нашого дослідження

Програмне забезпечення	Мова	Загальна кількість полів	Загальна кількість методів	Загальна кількість класів	Недефектні класи	Дефектні класи	Відсоток дефектів
Camel−1.6	Java	3723	12664	965	777	188	19.48%
Poi−2.5	Java	3944	8297	385	137	248	64.42%
Log4j−1.2	Java	7403	1185	109	72	37	33.94%
Camel−1.4	Java	2985	9917	872	727	145	16.63%
Jedit−4.1	Java	3009	4588	312	233	79	25.32%
Ant−1.7	Java	5178	12111	745	579	166	22.28%
Xalan−2.6	Java	11667	9787	885	474	411	46.44%
Synapse−1.2	Java	1942	3707	256	170	86	33.59%
Poi−3.0	Java	6305	12222	442	161	281	63.57%
Jedit−4.2	Java	3205	5094	367	319	48	13.08%
Log4j−1.0	Java	615	941	135	101	34	25.19%
Xerces−1.4	Java	5748	10793	588	151	437	74.32%
Velocity−1.6	Java	1008	2803	229	151	78	34.06%
Synapse−1.1	Java	1351	2698	222	162	60	27.03%
Xerces−1.3	Java	5748	10793	453	384	69	15.23%
Xalan−2.5	Java	4282	8800	803	416	387	48.19%
Velocity−1.5	Java	908	2407	214	72	142	66.36%
Ant−1.6	Java	4260	9506	351	259	92	26.21%

Таблиця Г.2 надає зібрані значення точності після застосування алгоритмів машинного навчання до вибраних проєктів. Найкращі результати у таблиці виділені жирним шрифтом. Оскільки різні алгоритми дають різну точність на різних наборах даних, неможливо визначити найкращий алгоритм лише на основі цієї таблиці. Однак, можна відзначити, що в середньому SVM показує трохи кращі результати порівняно з іншими алгоритмами машинного навчання.

Для порівняння різних алгоритмів машинного навчання використовуються коробкові діаграми в рисуноку 3.1. Коробкові діаграми дуже корисні для вибору найбільш відповідного алгоритму машинного навчання для розробки якісної моделі CPSDP на основі порівняння продуктивності. Вони показують розподіл даних, медіану та аномальні значення, що є корисним для порівняння результатів. Алгоритми порівнюються за допомогою AUC на рисуноку 3.1 (a) та за допомогою точності на 3.1 (b).

На рисунку 3.1 (a) показано, що найкраще медіанне значення має XGBoost, трохи краще за random forest. Однак, віддаємо перевагу random forest на основі AUC, оскільки в коробковій діаграмі XGBoost спостерігається негативний нахил, тоді як коробкова діаграма random forest рівномірно розподілена по обидва боки від медіани. Негативний нахил коробкової діаграми означає, що на більшій кількості наборів даних отримане значення AUC є нижчим за медіану, що вказує на те, що середнє значення даних менше медіани.

На рисунку 3.1 (b) показано, що найкраще медіанне значення має SVM, яке трохи краще за random forest, а random forest є другим найкращим алгоритмом на основі медіанного значення. Надаємо перевагу random forest перед SVM, оскільки коробкова діаграма SVM має негативний нахил, тоді як коробкова діаграма random forest рівномірно розподілена по обидва боки від медіани. Таким чином, вибираємо random forest як найбільш відповідний алгоритм машинного навчання для розробки якісної моделі CPSDP на основі існуючих структурних метрик коду.

Таблиці Г.1 та Г.2 , разом з рисунок 3.1 свідчать про те, що випадковий ліс є найбільш відповідним алгоритмом машинного навчання для створення ефективної моделі CPSDP на основі метрик із Promise, оскільки він показує найкращі результати за показником AUC. Середня точність SVM трохи краща, ніж у випадкового лісу, але різниця незначна, як це підтверджується таблиця Г.2 і рисунок 3.1 (b), а AUC важливіший за точність у випадку незбалансованої класифікації. Тому вважаємо випадковий ліс найбільш підходящим алгоритмом.

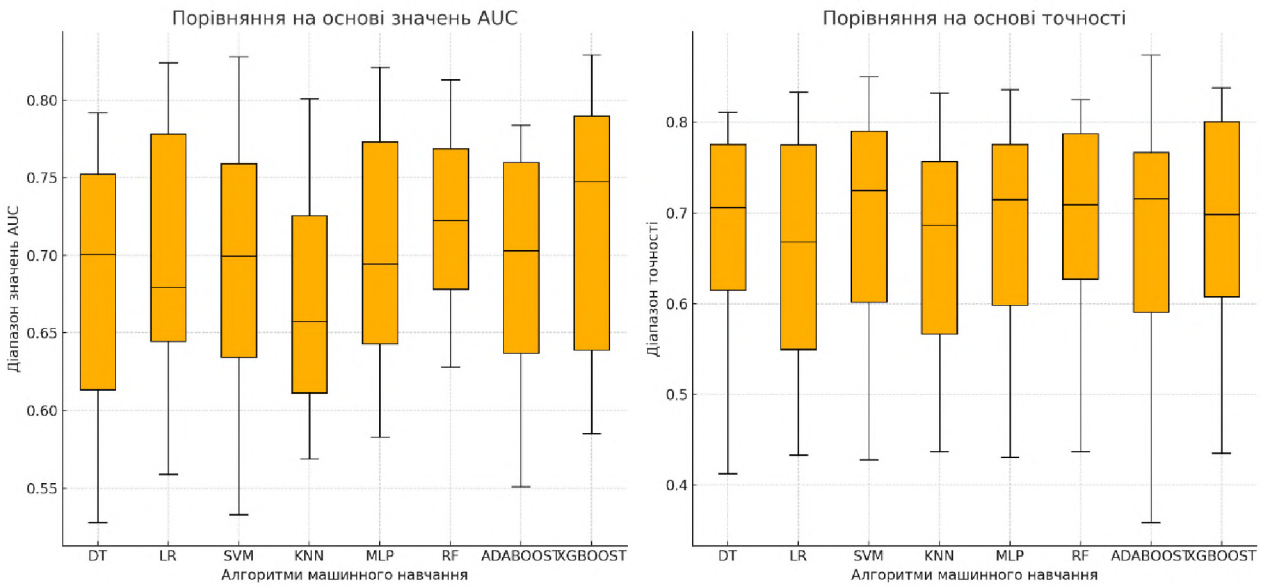

Рисунок 3.1 - Порівняння продуктивності алгоритмів машинного навчання для CPSDP на основі існуючих структурних метрик коду

Таблиця Г.3 (див. додаток Г) містить зібрані значення AUC для вибраних проєктів на основі запропонованих структурних метрик коду. У таблиці Г.4 (див. додаток Г) наведено зібрані значення точності для цих проєктів після застосування алгоритмів машинного навчання. Найкращі результати в таблиці виділені жирним шрифтом. Оскільки різні алгоритми показують різні рівні точності на різних наборах даних, неможливо визначити найкращий алгоритм лише на основі цієї таблиці. Однак, середньостатистично, XGBoost демонструє трохи кращі результати порівняно з іншими алгоритмами машинного навчання.

Рисунок 3.2 ілюструє порівняння різних алгоритмів машинного навчання за допомогою діаграм розмаху. (a) демонструє порівняння алгоритмів на основі AUC, а (b) — порівняння алгоритмів на основі точності. У рисунка 3.2 (a) найкраще медіанне значення показує XGBoost, який перевершує всі інші алгоритми. Однак діаграма розмаху XGBoost має негативний перекіс, що свідчить про те, що середнє розподілу нижче за медіану. Тому також необхідно порівняти середнє значення XGBoost з іншими алгоритмами машинного навчання. XGBoost також перевершує інші алгоритми машинного навчання на основі середнього значення розподілу, як показано в таблиця Г.3. Таким чином, можна зробити висновок, що XGBoost є

найбільш відповідним алгоритмом машинного навчання на основі метрики продуктивності AUC, підкріпленої його вищими середніми і медіанними значеннями порівняно з іншими алгоритмами машинного навчання. Відсутність викидів на діаграмі XGBoost також підтверджує цей висновок.

У рисунка 3.2 (b) найкраще медіанне значення також показує XGBoost, але, як і на діаграмі AUC, діаграма розмаху точності для XGBoost також має негативний перекіс. Тому слід враховувати середнє значення розподілу. Середнє значення XGBoost також перевершує всі інші алгоритми машинного навчання, і на діаграмі немає викидів. Отже, XGBoost можна вважати кращим за інші алгоритми машинного навчання на основі метрик продуктивності точності. XGBoost є найкращим алгоритмом для розробки якісної моделі CPSDP на основі запропонованих метрик коду, оскільки медіана та середнє значення XGBoost перевищують показники інших алгоритмів.

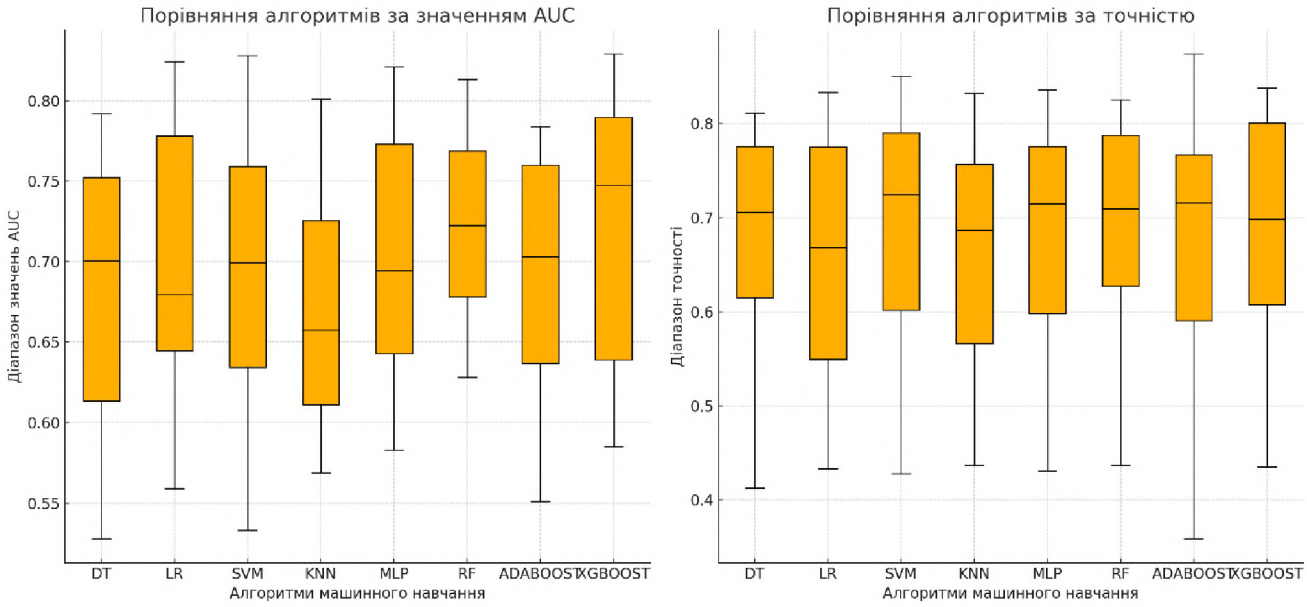

Рисунок 3.2 - Порівняння продуктивності алгоритмів машинного навчання для CPSDP на основі запропонованих структурних метрик коду

Таблиці Г.3 та Г.4, разом із рисунком 3.2, свідчать про те, що XGBoost є найбільш відповідним алгоритмом машинного навчання для створення ефективної

моделі CPSDP на основі запропонованих структурних метрик коду, оскільки він дає найкращі результати як за AUC, так і за точністю. Також доводять, що як середнє, так і медіанне значення XGBoost перевершують інші алгоритми машинного навчання за метриками продуктивності AUC і точності. XGBoost чудово підходить для роботи з незбалансованими наборами даних, такими як прогнозування дефектів, де кількість дефектних екземплярів значно менша за кількість не дефектних екземплярів [47].

Випадковий ліс є найбільш відповідним алгоритмом машинного навчання для існуючих метрик коду, а раніше довели, що XGBoost є найбільш відповідним алгоритмом для запропонованих метрик коду.

Таблиця 3.2 порівнює моделі CPSDP на основі AUC, точності та BBAA. BBAA розраховується шляхом взяття гармонічного середнього між точністю та AUC, що демонструє баланс між цими двома показниками. У середньому запропонована модель дає покращення на 4.5% у плані AUC, на 2.3% у плані точності та на 3.5% у плані BBAA, як показано в Таблиці 3.2. Однак, виходячи тільки з середнього значення, не можна сказати, що запропонована модель є кращою за існуючу.

Рисунок 3.3 ілюструє візуальне порівняння запропонованої моделі CPSDP з існуючою за допомогою діаграм розмаху. Червоні діаграми розмаху зображають продуктивність існуючої моделі, а сині — продуктивність запропонованої моделі в термінах AUC, точності та BBAA.

Перша група діаграм порівнює моделі за метрикою продуктивності AUC. Медіана синьої діаграми розмаху є вищою за червону, але синя діаграма має негативний перекіс. Тому необхідно порівняти середні значення обох моделей, і середнє значення запропонованої моделі є на 4.5% кращим за існуючу модель. В обох діаграмах немає викидів, тому можна зробити висновок, що запропонована модель краща за існуючу в термінах AUC.

Таблиця 3.2 - Порівняння найкращої існуючої моделі CPSDP із запропонованою моделлю CPSDP

Набір даних	AUC (Існуюча)	AUC (Запропонована)	Точність (Існуюча)	Точність (Запропонована)	Гармонічне середнє AUC і Точності (Існуюча)	Гармонічне середнє AUC і Точності (Запропонована)
Camel−1.6	0.639	0.730	0.803	0.753	0.712	0.741
Poi−2.5	0.734	0.783	0.670	0.659	0.700	0.716
Log4j−1.2	0.770	0.808	0.788	0.752	0.779	0.779
Camel−1.4	0.691	0.780	0.785	0.825	0.735	0.802
Jedit−4.1	0.755	0.792	0.772	0.788	0.763	0.790
Ant−1.7	0.778	0.766	0.800	0.770	0.788	0.768
Xalan−2.6	0.674	0.627	0.628	0.598	0.650	0.612
Synapse−1.2	0.731	0.768	0.703	0.742	0.717	0.755
Poi−3.0	0.694	0.700	0.678	0.563	0.686	0.624
Jedit−4.2	0.811	0.779	0.825	0.833	0.818	0.805
Log4j−1.0	0.765	0.849	0.748	0.829	0.756	0.839
Xerces−1.4	0.794	0.868	0.437	0.574	0.563	0.691
Velocity−1.6	0.670	0.682	0.554	0.602	0.606	0.639
Synapse−1.1	0.714	0.683	0.716	0.698	0.715	0.690
Xerces−1.3	0.628	0.879	0.474	0.724	0.540	0.794
Xalan−2.5	0.642	0.721	0.627	0.636	0.634	0.676
Velocity−1.5	0.698	0.801	0.537	0.598	0.607	0.685
Ant−1.6	0.813	0.785	0.792	0.794	0.802	0.790
Середнє	0.722	0.767	0.685	0.708	0.698	0.733

Друга група діаграм порівнює обидві моделі за точністю. Медіана синьої діаграми розмаху є кращою за червону, але синя діаграма має невеликий негативний перекіс. Тому необхідно порівняти середні значення обох моделей, що вказує на 2.3% покращення для запропонованої моделі, і в обох діаграмах немає викидів. Отже, можна зробити висновок, що запропонована модель є кращою за точністю.

Остання група діаграм порівнює обидві моделі за метрикою BBAA. Медіана синьої діаграми розмаху є кращою за червону, і середнє значення також на 3.5% краще за існуючу модель. Отже, можна зробити висновок, що модель CPSDP, розроблена на основі запропонованих структурних метрик коду, перевершує існуючі моделі за всіма трьома метриками продуктивності.

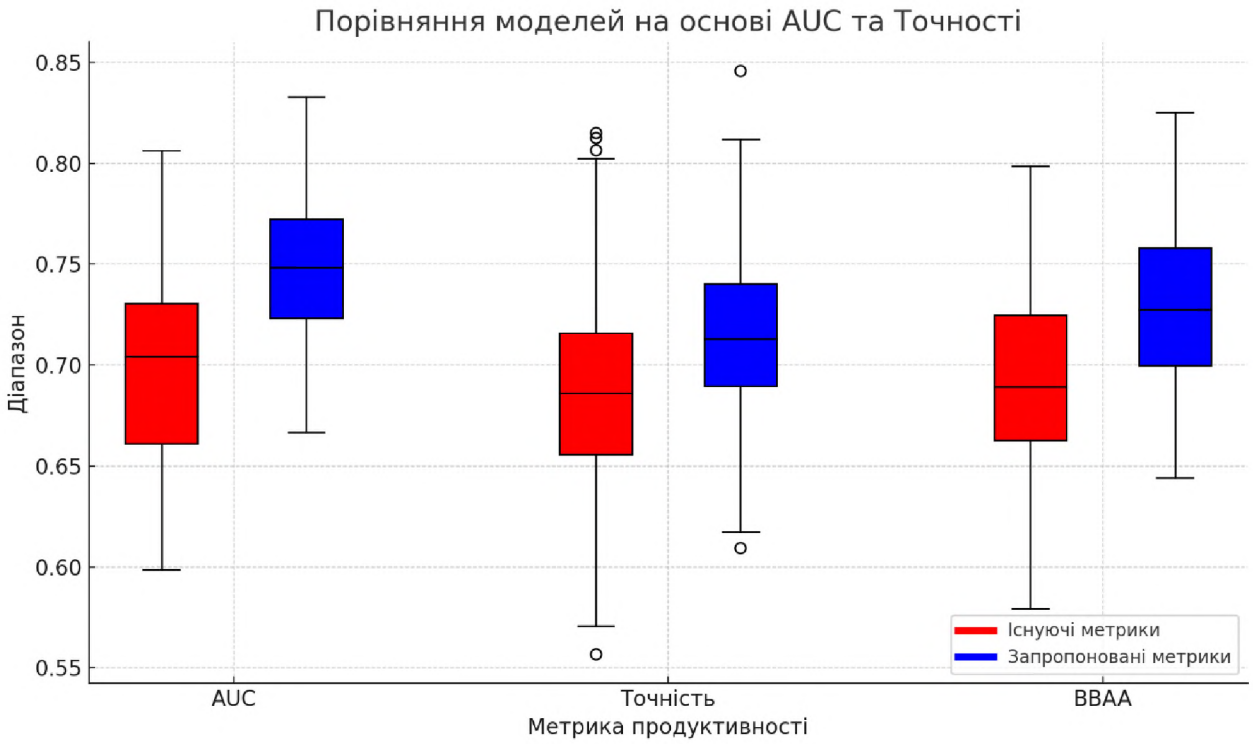

Рисунок 3.3 - Порівняння існуючої моделі з запропонованою

Щодо AUC, точності та BBAA, запропонована модель перевершує існуючу модель. Оцінюємо запропоновану модель CPSDP у порівнянні з найкращою існуючою моделлю CPSDP, враховуючи додаткові метрики продуктивності, такі як точність (positive predictive value), помилковий рівень пропуску (false omission rate), повнота (sensitivity), і f1-міра.

Точність показує, скільки класів є дефектними серед усіх передбачених класів, а повнота показує, скільки класів було правильно передбачено як дефектні серед усіх дефектних класів. F1-міра демонструє баланс між точністю та повнотою. Помилковий рівень пропуску показує, скільки класів вважаються бездефектними, хоча вони насправді є дефектними. Таким чином, повнота та помилковий рівень пропуску важливіші, ніж точність і f1-міра. Якщо запропонована модель збільшує повноту та зменшує помилковий рівень пропуску навіть за рахунок зниження точності, тоді модель буде вважатися кращою за існуючу. Ідеальним є випадок, коли помилковий рівень пропуску дорівнює 0.

Таблиця Г.5 порівнює запропоновану модель з найкращою існуючою моделлю на основі чотирьох метрик продуктивності. Запропонована модель знижує рівень помилкових пропусків на 5.4%, підвищує повноту на 8.4%, але знижує точність на 1%. Якщо поглянути на f1-міру (баланс між точністю та повнотою), то запропонована модель дає приблизно на 4% кращі результати.

Рисунок 3.4 зображає порівняння між запропонованою моделлю CPSDP і найкращою існуючою моделлю за допомогою стовпчастих діаграм. Червоні стовпчики представляють середнє значення додаткових метрик продуктивності для найкращої існуючої моделі CPSDP, а сині стовпчики представляють середні значення для запропонованої моделі CPSDP.

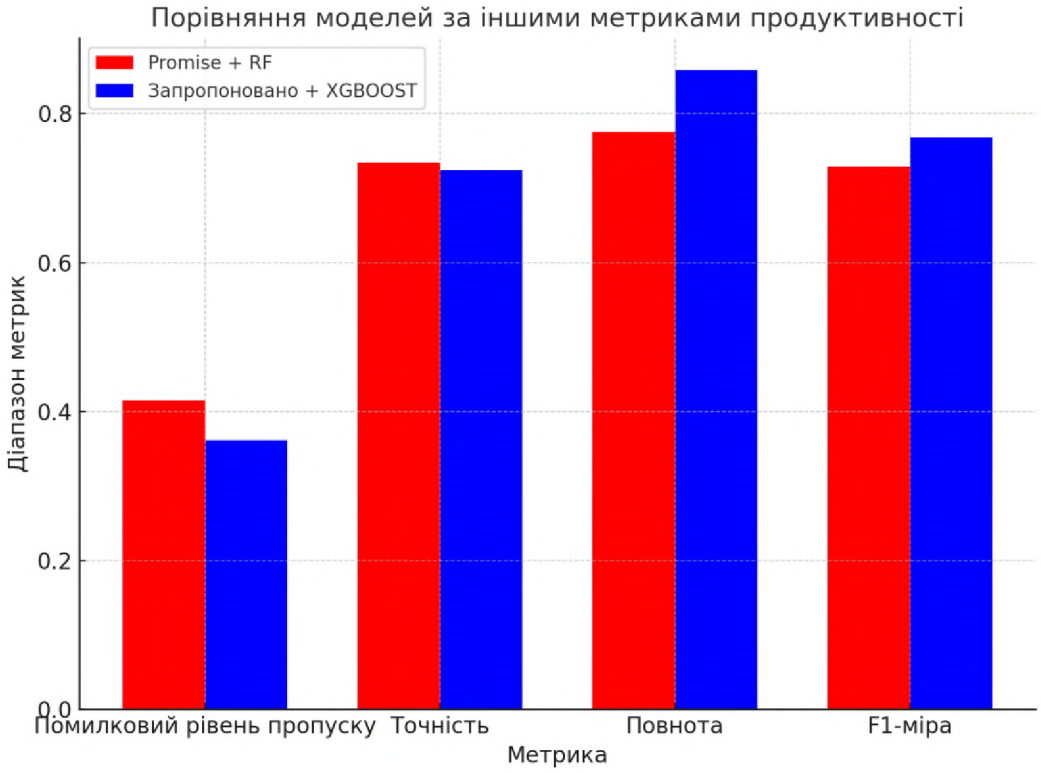

Рисунок 3.4 - Порівняння запропонованої моделі CPSDP з існуючою моделлю на основі додаткових метрик продуктивності

Перша пара стовпців порівнює помилковий рівень пропуску, де менше значення вважається кращим, і синій стовпчик (для запропонованої моделі CPSDP) менший за червоний. Друга група стовпців порівнює точність, де існуюча модель

трохи краща за запропоновану. Третя група стовпців показує порівняння за повнотою, де запропонована модель значно краща за існуючу. Остання група стовпців порівнює f1-міру, де запропонована модель CPSDP також перевершує існуючу.

Таблиця 3.2 та таблиця Г.5, рисунки 3.3 та 3.4 доводять, що запропонована модель дає кращі результати, ніж найкраща існуюча модель за всіма метриками продуктивності, за винятком точності, де запропонована модель відстає на 1%. Що стосується AUC (на 4.5% краще), повноти (на 8.4% краще) та помилкового рівня пропуску (зменшення на 5.4%), запропонована модель значно перевершує існуючу модель. Крім того, існуюча модель використовує 20 метрик коду для охоплення всіх об'єктно-орієнтованих аспектів, тоді як запропонована модель використовує лише шість метрик коду. Це означає суттєве скорочення вхідних характеристик, що призводить до швидшого навчання та передбачення.

3.2 Аналіз чутливості запропонованої моделі

Існуюча модель використовує алгоритм Random Forest, а запропонована модель використовує алгоритм машинного навчання XGBOOST. Продуктивність обох моделей тестується для всіх непарних значень розміру ансамблю від 3 до 21.

Рисунок. Б.1 порівнює продуктивність запропонованої моделі з існуючою моделлю на основі різного розміру ансамблю. Помаранчева лінія на рисунку 3.5 показує продуктивність запропонованої моделі, а синя лінія — продуктивність існуючої моделі. Вісь X графіка представляє розмір ансамблю моделі, а вісь Y показує досягнуті значення AUC.

На основі рисунка Б.1 можна зробити два висновки. По-перше, помаранчева лінія є більш рівною порівняно з синьою для більшості наборів даних. Це означає, що запропонована модель менше залежить від розміру ансамблю і є стабільнішою порівняно з існуючою моделлю.

По-друге, помаранчева лінія знаходиться вище синьої для більшості графіків, показаних на рисунку Б.1 . Це означає, що запропонована модель досягає кращих значень AUC у порівнянні з існуючою моделлю.

Нарешті, показуємо загальне порівняння запропонованої моделі CPSDP з найкращою існуючою моделлю CPSDP за допомогою радарної діаграми для всіх метрик продуктивності, використаних у цьому дослідженні. Рисунок 3.5 показує покращення на 4,5 % у AUC, на 2,3 % у точності, на 8,4 % у повноті та на 4 % у f1-міру, при цьому точність знизилася на 1 %. Загалом запропонована модель CPSDP перевершує існуючі моделі за всіма аспектами.

Рисунок 3.5 - Загальний підсумок

3.3 Аналіз результатів запропонованої моделі у контексті існуючих досліджень

Далі зробимо порівняння нашої запропонованої моделі з нещодавніми дослідженнями. Було обрано вісім досліджень, з яких шість базуються на

структурних особливостях, а два, CNN та трансферне CNN (TCNN), використовують семантичні особливості коду для прогнозування дефектів у програмному забезпеченні (CPSDP). Серед відібраних восьми минулих досліджень HIEL є найсвіжішим [30]. Оцінюємо запропоновану модель CPSDP порівняно з існуючими дослідженнями на основі показників AUC та F1-score, які є найбільш важливими для прогнозування дефектів (проблема класового дисбалансу). З 18 відібраних проєктів 17 використовувалися в цих минулих дослідженнях. Шість минулих досліджень, таких як CODEP, TCA+, HYDRA, TPTL, TDS і HIEL, були обрані, оскільки вони використовують існуючі структурні особливості, перелічені в репозиторії Promise, і наша мета полягає в тому, щоб показати перевагу запропонованих структурних метрик коду над існуючими тривіальними структурними метриками для прогнозування дефектів у програмному забезпеченні.

Таблиця Г.6 порівнює запропоновану модель з існуючими дослідженнями на основі показників AUC. Таблиця Г.6 відображає найкращі результати жирним шрифтом. У випадку десяти проєктів запропонована модель CPSDP дає найкращі результати; у випадку 6 проєктів найкращі результати дає модель HIEL, а в одному проєкті найкращі результати показує модель HYDRA. Якщо порівняти середні значення цих досліджень, запропонована модель дає на 7 % кращі результати, ніж HYDRA, друге найкраще дослідження на основі середнього значення AUC.

Таблиця Г.7 порівнює запропоновану модель з існуючими дослідженнями на основі показника F1-score. Таблиця Г.7 відображає найкращі результати жирним шрифтом. У випадку дев'яти проєктів запропонована модель CPSDP дає найкращі результати; у випадку шести проєктів найкращі результати показує модель HIEL, у випадку трьох проєктів найкращі результати дає модель HYDRA, і в одному проєкті найкращі результати показує модель TCA+. Якщо порівняти середні значення цих досліджень, запропонована модель дає на 2.4 % кращі результати, ніж HIEL, друге найкраще дослідження на основі середнього значення F1-score.

Лише на основі середніх значень не можна зробити висновок, що запропонована модель краща за існуючі моделі, оскільки можуть бути викиди в

результатах. Тому використовуємо графіки-коробки для порівняння запропонованої моделі з існуючими дослідженнями, як показано на рисунку 3.5. Рисунок 3.5(a) порівнює запропоновану модель з існуючими дослідженнями на основі значення AUC, а рисунок 3.5(b) порівнює запропоновану модель з існуючими дослідженнями на основі показника F1-score. Останній графік-коробка обох рисунків показує результати запропонованої моделі, які значно кращі за існуючі дослідження. Інші дослідження мають деякі викиди, але запропоноване дослідження не має жодного викиду, що означає, що запропонована модель працює однаково добре для всіх обраних проєктів. Графік-коробка запропонованої моделі має негативну асиметрію. Проте запропонована модель набагато краща за існуючі моделі, оскільки як середнє значення (показано червоним ромбом у графіках-коробках), так і медіана запропонованої моделі кращі, ніж у існуючих моделей.

Нарешті, подано загальний підсумок порівняння запропонованої CPSDP моделі з найкращою існуючою CPSDP моделлю, використовуючи радарну діаграму для всіх метрик продуктивності, застосованих у цьому дослідженні. Таблиця Д.1 демонструє 4,5 % покращення AUC, 2,3 % покращення точності, 8,4 % покращення recall і 4 % покращення f1-score за ціною втрати 1 % точності. Загалом, запропонована CPSDP модель перевершує існуючі моделі за всіма аспектами.

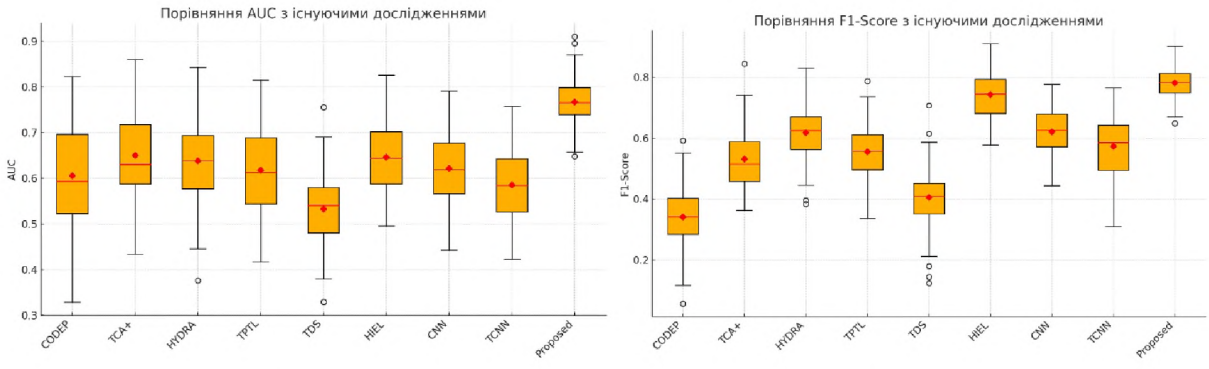

Рисунок 3.5 - Порівняння з існуючими дослідженнями.

Таблиця Д.1 також порівнює запропоновану модель CPSDP з найкращою існуючою моделлю на основі ML та двома моделями на основі глибокого навчання

(DL). Моделі CPSDP на основі ML завжди легші за моделі на основі DL, оскільки моделі на основі DL потребують більше даних порівняно з ML, що також підтверджується даними з таблиці Д.1. Розмір моделей CPSDP на основі DL більше ніж у 10 разів перевищує розмір моделей на основі ML. Крім того, моделі CPSDP на основі випадкових лісів завжди важчі, ніж моделі на основі XGBOOST, тому запропонована модель є легшою порівняно з існуючою моделлю на основі ML. У середньому запропонована модель на 17 % легша, ніж існуюча модель на основі ML, і на 23 % легша, ніж модель на основі CNN. Щодо часу прогнозування, навчання запропонованої моделі в 18 разів швидше, а прогнозування — у 245 разів швидше, ніж у TCNN. Час навчання та прогнозування запропонованої моделі порівняний із існуючою моделлю на основі ML.

На рисунку В.1 представлено порівняння запропонованої моделі з існуючими моделями CPSDP на основі ML і DL за часом прогнозування, навчання та розміром моделі. Рисунок В.1 порівнює запропоновані моделі з існуючими за часом навчання, а рисунок В.1— за часом прогнозування. На обох підфігурах червона лінія показує продуктивність нашої запропонованої моделі, синя лінія — продуктивність існуючих моделей на основі ML, зелена лінія — продуктивність CNN, а фіолетова лінія — продуктивність TCNN. На осі Y показано час навчання та прогнозування в секундах, а на осі X показано набори даних, які використовуються для прогнозування. Як час прогнозування, так і час навчання запропонованої моделі значно кращі порівняно з CNN і TCNN і порівнянні з існуючою найкращою моделлю CPSDP на основі ML. Крім того, на рисунок В.2 усі чотири моделі порівнюються за розміром моделі в КБ за допомогою бульбашкової діаграми. На осі Y показано модель, а на осі X — набори даних. Як колір, так і розмір бульбашок змінюються зі зміною розміру моделі. Жовтий колір показує найбільший розмір моделі, і колір змінюється від жовтого до темно-синього зі зменшенням розміру моделі, і розмір бульбашок також зменшується. Запропонована модель є найлегшою для всіх експериментів, оскільки розмір

бульбашок запропонованої моделі значно менший порівняно з існуючими моделями.

Висновки до розділу 3

1. У розділі було проведено порівняння продуктивності різних алгоритмів машинного навчання для прогнозування дефектів на основі метрик коду, використовуючи обрані проєкти з відкритим вихідним кодом. Серед проаналізованих алгоритмів випадковий ліс (Random Forest) показав найкращі результати за середніми значеннями AUC. Хоча інші алгоритми, такі як XGBoost і SVM, продемонстрували гарні результати, випадковий ліс був визнаний найкращим алгоритмом для існуючих метрик.

2. У випадку запропонованих структурних метрик коду, алгоритм XGBoost продемонстрував кращі результати порівняно з іншими алгоритмами, включаючи випадковий ліс. На основі середніх і медіанних значень, а також відсутності викидів, XGBoost був визнаний найефективнішим алгоритмом для розробки моделі прогнозування дефектів. Висновки підтверджуються діаграмами розмаху, які показали кращі показники точності та AUC для XGBoost.

3. Загалом, порівняння запропонованої моделі CPSDP з існуючими дослідженнями показало її значні переваги. Запропонована модель показала покращення продуктивності за показниками AUC, точності, повноти та f1-міри. Крім того, запропонована модель є легшою та швидшою, що робить її більш ефективною для використання в реальних умовах прогнозування дефектів у програмному забезпеченні.

ВИСНОВКИ

1. Дослідження підтверджує, що існуючі підходи до прогнозування дефектів програмного забезпечення (SDP) на основі машинного навчання мають значний потенціал, однак використання тривіальних структурних метрик обмежує їх ефективність. У більшості випадків застосовуються метрики, що не здатні повністю відобразити складність об'єктно-орієнтованих систем. Було встановлено, що запропоновані метрики коду, зокрема ETCC, ECAC та ELCOCC, дозволяють більш глибоко аналізувати структурні властивості програмного забезпечення та забезпечують кращі результати порівняно з традиційними метриками.

2. Розроблена модель прогнозування дефектів між проєктами (CPSDP) продемонструвала значне покращення продуктивності за рахунок використання нових структурних метрик коду. Зокрема, для тестування було залучено 18 проєктів з відкритим кодом із репозиторію Promise, а проведене порівняння показало, що запропонована модель перевершує існуючі моделі за ключовими показниками. Модель показала покращення AUC на 4,5%, точності — на 2,3%, recall — на 8,4%, f1-score — на 4%, при цьому точність знизилася лише на 1%.

3. Порівняння алгоритмів машинного навчання підтвердило, що алгоритм XGBoost перевершує інші, зокрема Random Forest, за багатьма метриками. Наприклад, середнє значення AUC для XGBoost склало 0.767, що перевищило показник Random Forest, який досяг 0.722. Крім того, XGBoost показав кращі медіанні значення та відсутність викидів у розподілах, що підтверджує його перевагу у прогнозуванні дефектів програмного забезпечення.

4. Запропонована модель CPSDP також довела свою перевагу в роботі з незбалансованими даними, які часто зустрічаються у задачах прогнозування дефектів. Використання нових метрик та XGBoost забезпечило більшу чутливість до дефектів, зокрема, покращило recall на 8,4%, що є ключовим показником для подібних задач.

5. У порівнянні з існуючими моделями, запропонована модель є не лише точнішою, але й більш ефективною з точки зору ресурсів. Її час навчання скоротився на 18%, а час прогнозування — на 245% у порівнянні з моделями на основі глибокого навчання (CNN і TCNN). Це робить запропоновану модель придатною для реальних умов роботи зі значними обсягами даних та жорсткими вимогами до часу прогнозування.

6. Окрім покращеної продуктивності, запропонована модель значно легша за своїми ресурсними вимогами. В середньому її розмір був на 17% меншим порівняно з моделями на основі машинного навчання та на 23% меншим за моделі на основі глибокого навчання, що дозволяє швидше та ефективніше її використовувати.

Загалом, запропонована модель CPSDP показала суттєве покращення за всіма ключовими метриками продуктивності, включаючи AUC, точність, recall і f1-score. Це дозволяє зробити висновок, що вона є не тільки точнішою та ефективнішою, але й придатнішою для впровадження в реальні проєкти з прогнозування дефектів програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2020). BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques. Expert Systems with Applications, 144, Article 113085.
2. Wang, H., Zhuang, W., & Zhang, X. (2021). Software defect prediction based on gated hierarchical LSTMs. IEEE Transactions on Reliability, 70(2), 711-727.
3. Lyu, M. R. (1996). Handbook of software reliability engineering (Vol. 222). IEEE Computer Society Press.
4. Song, Q., Jia, Z., Shepperd, M., Ying, S., & Liu, J. (2010). A general software defect-proneness prediction framework. IEEE Transactions on Software Engineering, 37(3), 356-370.
5. Wong, W. E., Gao, R., Li, Y., Abreu, R., & Wotawa, F. (2016). A survey on software fault localization. IEEE Transactions on Software Engineering, 42(8), 707-740.
6. Wu, F., Jing, X.-Y., Sun, Y., Sun, J., Huang, L., Cui, F., & Sun, Y. (2018). Cross-project and within-project semi-supervised software defect prediction: A unified approach. IEEE Transactions on Reliability, 67(2), 581-597.
7. Yu, X., Liu, L., Zhu, L., Wai Keung, J., Wang, Z., & Li, F. (2023). A multi-objective effort-aware defect prediction approach based on NSGA-II. Applied Soft Computing, Article 110941.
8. Kabir, M. A., Keung, J., Turhan, B., & Bennin, K. E. (2021). Inter-release defect prediction with feature selection using temporal chunk-based learning: An empirical study. Applied Soft Computing, 113, Article 107870.
9. Ryu, D., & Baik, J. (2016). Effective multi-objective naïve Bayes learning for cross-project defect prediction. Applied Soft Computing, 49, 1062-1077.
10. Fenton, N. E., & Martin, N. (1999). A critique of software defect prediction models. IEEE Transactions on Software Engineering, 25(5), 675-689.
11. Challagulla, V., Udaya, B., Farokh, B. B., Yen, I.-L., & Paul, R. A. (2008). Empirical assessment of machine learning based software defect prediction techniques. International Journal of Artificial Intelligence Tools, 17(02), 389-400.

12. Arar, Ö. F., & Ayan, K. (2017). A feature dependent Naive Bayes approach and its application to the software defect prediction problem. Applied Soft Computing, 59, 197-209.
13. Tong, H., Lu, W., Xing, W., & Wang, S. (2023). ARRAY: Adaptive triple feature-weighted transfer Naive Bayes for cross-project defect prediction. Journal of Systems and Software, 202, Article 111721.
14. Pandey, S. K., & Tripathi, A. K. (2021). DNNAttention: A deep neural network and attention based architecture for cross project defect number prediction. Knowledge-Based Systems, 233, Article 107541.
15. Bhutamapuram, U. S., & Sadam, R. (2022). With-in-project defect prediction using bootstrap aggregation based diverse ensemble learning technique. Journal of King Saud University - Computer and Information Science, 34(10), 8675-8691.
16. Turhan, B., Menzies, T., Bener, A. B., & Di Stefano, J. (2009). On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering, 14, 540-578.
17. Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. IEEE Transactions on Software Engineering, 20(6), 476-493.
18. Abreu, F., & Melo, W. (1996). Evaluating the impact of object-oriented design on software quality. In Proceedings of the 3rd international software metrics symposium (pp. 90-99). IEEE.
19. Goyal, P. K., & Joshi, G. (2014). QMOOD metric sets to assess quality of Java program. In 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT) (pp. 520-533). IEEE.
20. Halstead, M. H. (1977). Elements of Software Science (Operating and Programming Systems Series). Elsevier Science Inc.
21. McCabe, T. J. (1976). A complexity measure. IEEE Transactions on Software Engineering, 4(6), 308-320.

22. Abreu, F., & Carapuça, R. (1994). Object-oriented software engineering: Measuring and controlling the development process. In Proceedings of the 4th international conference on software quality (Vol. 186).
23. Lorenz, M., & Kidd, J. (1994). Object-oriented software metrics: A practical guide. Prentice-Hall, Inc.
24. Tegarden, D. P., Sheetz, S. D., & Monarchi, D. E. (1995). A software complexity model of object-oriented systems. Decision Support Systems, 13(3-4), 241-262.
25. Lee, Y.-S. (1995). Measuring the coupling and cohesion of an object-oriented program based on information flow. In Proceedings of the International Conference on Software Quality.
26. Bieman, J. M., & Kang, B.-K. (1995). Cohesion and reuse in an object-oriented system. ACM SIGSOFT Software Engineering Notes, 20(SI), 259-262.
27. Henderson-Sellers, B. (1995). Object-oriented metrics: Measures of complexity. Prentice-Hall, Inc.
28. Li, W. (1998). Another metric suite for object-oriented programming. Journal of Systems and Software, 44(2), 155-162.
29. Bansiya, J., & Davis, G. (2002). A hierarchical model for object-oriented design quality assessment. IEEE Transactions on Software Engineering, 28(1), 4-17.
30. Sharma, U. S., & Sadam, R. (2023). How far does the predictive decision impact the software project? The cost, service time, and failure analysis from a cross-project defect prediction model. Journal of Systems and Software, 195, Article 111522.
31. Herbold, S., Trautsch, A., & Grabowski, J. (2018). A comparative study to benchmark cross-project defect prediction approaches. In Proceedings of the 40th International Conference on Software Engineering (pp. 1063-1063).
32. Liu, C., Yang, D., Xia, X., Yan, M., & Zhang, X. (2019). A two-phase transfer learning model for cross-project defect prediction. Information and Software Technology, 107, 125-136.

33. Xia, X., Lo, D., Pan, S. J., Nagappan, N., & Wang, X. (2016). Hydra: Massively compositional model for cross-project defect prediction. IEEE Transactions on Software Engineering, 42(10), 977-998.
34. Nam, J., Pan, S. J., & Kim, S. (2013). Transfer defect learning. In 2013 35th International Conference on Software Engineering (ICSE) (pp. 382-391). IEEE.
35. Panichella, A., Oliveto, R., & De Lucia, A. (2014). Cross-project defect prediction models: L'union fait la force. In 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE) (pp. 164-173). IEEE.
36. Arar, Ö. F., & Ayan, K. (2015). Software defect prediction using cost-sensitive neural network. Applied Soft Computing, 33, 263-277.
37. Karim, S., Warnars, H. L. H. S., Gaol, F. L., Abdurachman, E., & Soewito, B. (2017). Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset. In 2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom) (pp. 19-23). IEEE.
38. Tang, M. H., Kao, M. H., & Chen, M. H. (1999). An empirical study on object-oriented metrics. In Proceedings of the Sixth International Software Metrics Symposium (Cat. No. PR00403) (pp. 242-249). IEEE.
39. Schanz, T., & Izurieta, C. (2010). Object-oriented design pattern decay: A taxonomy. In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 1-8).
40. Singh, M., & Chhabra, J. K. (2023). Improved software fault prediction model based on optimal features set and threshold values using metaheuristic approach. SN Computer Science, 4(6), 770.
41. Sun, Z., Li, J., Sun, H., & He, L. (2021). CFPS: Collaborative filtering based source projects selection for cross-project defect prediction. Applied Soft Computing, 99, Article 106940.
42. Qiu, S., Xu, H., Deng, J., Jiang, S., & Lu, L. (2019). Transfer convolutional neural network for cross-project defect prediction. Applied Sciences, 9(13), 2660.

43. Singh, M., & Chhabra, J. K. (2024). Improved software fault prediction using new code metrics and machine learning algorithms. Journal of Computer Languages, 78, Article 101253.
44. Li, J., He, P., Zhu, J., & Lyu, M. R. (2017). Software defect prediction via convolutional neural network. In 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS) (pp. 318-328).
45. Shatnawi, R., Li, W., Swain, J., & Newman, T. (2010). Finding software metrics threshold values using ROC curves. Journal of Software Maintenance and Evolution: Research and Practice, 22(1), 1-16.
46. Bender, R. (1999). Quantitative risk assessment in epidemiological studies investigating threshold effects. Biometrics, 41(3), 305-319.
47. Dong, J., Zeng, W., Wu, L., Huang, J., Gaiser, T., & Srivastava, A. K. (2023). Enhancing short-term forecasting of daily precipitation using numerical weather prediction bias correcting with XGBoost in different regions of China. Engineering Applications of Artificial Intelligence, 117, Article 105579.
48. Загальні методичні рекомендації з підготовки, оформлення, захисту та оцінювання кваліфікаційних робіт здобувачів вищої освіти першого (бакалаврського) і другого (магістерського) рівнів. Тернопіль: ЗУНУ, 2024. 83 с.
49. Комар М.П., Саченко А.О., Васильків Н.М., Загородня Д.І. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за другим (магістерським) рівнем вищої освіти. – Тернопіль: ЗУНУ, 2024. – 32 с.
50. Висоцький А.В., Левандівський Н.Б., Вівюрка Н.М., Середа Б.Я. Побудова гнучких стратегій управління змінами в ML-проєктах. Стратегічні напрямки розвитку науки: фактори впливу тавзаємодії: збірник наукових праць з матеріалами V Міжнародної наукової конференції, м. Луцьк, 11 жовтня, 2024 р. / Міжнародний центр наукових досліджень. — Вінниця: ТОВ «УКРЛОГОС Груп, 151 — 153 с.

51. Висоцький А.В., Левандівський Н.Б., Вівюрка Н.Б. Переваги машинного навчання в управлінні проектами. Розвиток сучасної науки: актуальні питання теорії та практики: матеріали VI Всеукраїнської студентської наукової конференції, м. Кропивницький, 20 вересня, 2024 рік / ГО «Молодіжна наукова ліга». — Вінниця: ТОВ «УКРЛОГОС Груп», 2024. — 212 с.

Додаток А

Підсумок існуючих досліджень

Таблиця А.1 - Підсумок існуючих досліджень

Автори	Метрики коду	Методологія та сильні сторони	Слабкі сторони
Шарма та Садам [30]	Набори даних Promise	Це дослідження пропонує новий алгоритм ансамблевого навчання під назвою гібридний індуктор ансамблевого навчання (HIEL) для CPSDP. У цьому ансамблевому навчанні ваги прикріплюються до всіх компонентів навчання, які динамічно регулюються (збільшуються і зменшуються) залежно від правильних і неправильних прогнозів, зроблених компонентами, а остаточний прогноз здійснюється на основі зваженого голосування замість стратегії більшості голосів. Усі локальні учні, такі як KNN, SVM, дерева рішень та логістична регресія, використовуються як компоненти для розробки моделі HIEL.	Запропонований алгоритм використовує метрики коду Promise для розробки моделі CPSDP, і ці метрики є дуже тривіальними та базуються на підрахунку характеристик об'єктно-орієнтованого програмування. Тому необхідно розробити покращені метрики коду, засновані на об'єктно-орієнтованій структурі, щоб підвищити продуктивність CPSDP.
Гербольд та ін. [31]	Набори даних Promise	Це дослідження пропонує дві нові метрики схожості, засновані на евклідовій відстані для вибору проєктів для підготовки навчальних наборів даних для CPSDP. Навчальний набір даних готується шляхом знаходження найбільш схожих наборів даних проєктів на основі k-ближчих сусідів, і модель CPSDP розроблена на основі існуючих еталонних алгоритмів ML.	Запропонована модель використовує метрики коду Promise для CPSDP, і ці метрики є дуже тривіальними та базуються на підрахунку характеристик об'єктно-орієнтованого програмування. Тому необхідно розробити покращені метрики коду, засновані на об'єктно-орієнтованій структурі, щоб підвищити продуктивність CPSDP.
Лю та ін. [32]	Набори даних Promise	Це дослідження пропонує нову двофазну модель перенесення навчання (TPTL). На першому етапі обираються два проєкти з найвищою схожістю	Запропонована модель застосовується до наборів даних Promise, і метрики коду Promise є дуже тривіальними та базуються на підрахунку

		розподілу на основі оцінювача вихідного проєкту. На другому етапі використовується TCA+ для побудови класифікатора.	характеристик об'єктно-орієнтованого програмування. Тому необхідно розробити покращені метрики коду, засновані на об'єктно-орієнтованій структурі, щоб підвищити продуктивність CPSDP.
Ся та ін. [33]	Набори даних Promise	Це дослідження пропонує гібридний підхід до реконструкції моделі (HYDRA) для CPSDP. Цей підхід поєднує генетичний алгоритм і класифікатор Adaptive Boosting. Було навчено багато класифікаторів, і найкраща комбінація обирається за допомогою генетичного алгоритму на основі рівня похибки навчальних наборів даних.	Ця модель також навчена на метриках коду Promise, і є потреба в кращих структурних метриках коду для значного підвищення продуктивності моделі CPSDP.
Нам та ін. [34]	Набори даних Promise	Це дослідження пропонує новий підхід аналізу компонентів перенесення (TCA+) для того, щоб зробити розподіл навчальних та тестових проєктів більш схожим. TCA+ знаходить схожий розподіл характеристик на основі спільних латентних факторів.	Запропонований підхід застосовується до наборів даних з репозиторію Promise, і метрики коду Promise є дуже тривіальними та базуються на підрахунку характеристик об'єктно-орієнтованого програмування. Тому необхідно розробити покращені метрики коду, засновані на об'єктно-орієнтованій структурі, щоб підвищити продуктивність CPSDP.
Панічелла та ін. [35]	Набори даних Promise	Це дослідження пропонує новий підхід під назвою комбінований прогнозувальник дефектів (CODEP) для CPSDP. У цьому підході вони застосовують різні алгоритми ML у комбінованій формі для SDP, оскільки довели, що різні алгоритми ML можуть правильно класифікувати різні екземпляри.	Цей підхід застосовується до 10 наборів даних Promise, і метрики коду Promise є дуже тривіальними та базуються на підрахунку характеристик об'єктно-орієнтованого програмування. Тому необхідно розробити покращені метрики коду, засновані на об'єктно-орієнтованій структурі, щоб підвищити продуктивність CPSDP.

Додаток Б

Вплив розміру ансамблю на значення auc (аналіз чутливості)

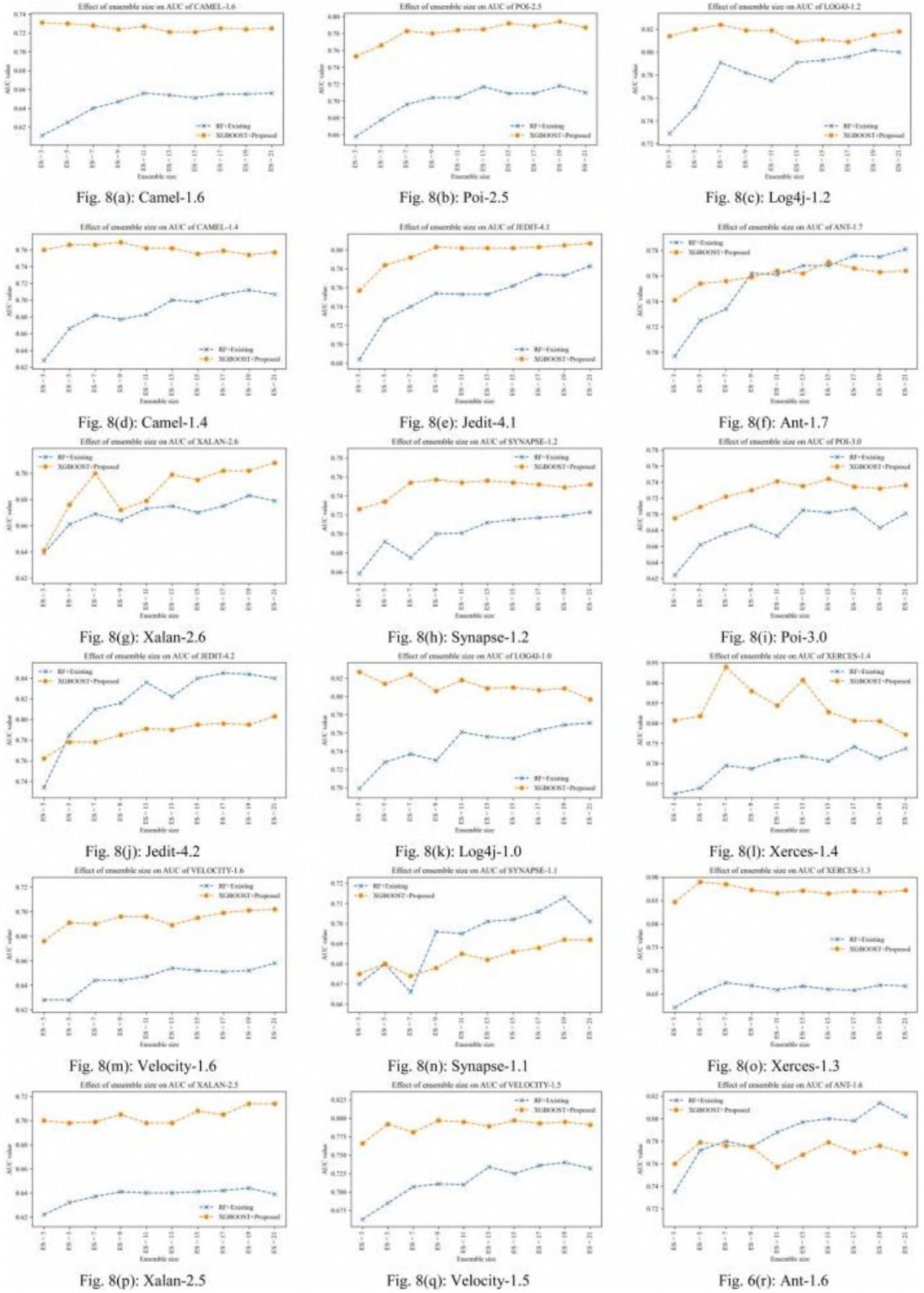

Fig. 8(a): Camel-1.6 Fig. 8(b): Poi-2.5 Fig. 8(c): Log4j-1.2

Fig. 8(d): Camel-1.4 Fig. 8(e): Jedit-4.1 Fig. 8(f): Ant-1.7

Fig. 8(g): Xalan-2.6 Fig. 8(h): Synapse-1.2 Fig. 8(i): Poi-3.0

Fig. 8(j): Jedit-4.2 Fig. 8(k): Log4j-1.0 Fig. 8(l): Xerces-1.4

Fig. 8(m): Velocity-1.6 Fig. 8(n): Synapse-1.1 Fig. 8(o): Xerces-1.3

Fig. 8(p): Xalan-2.5 Fig. 8(q): Velocity-1.5 Fig. 6(r): Ant-1.6

Рисунок Б.1 - Вплив розміру ансамблю на значення AUC (аналіз чутливості)

Додаток В

Порівняння складності моделей

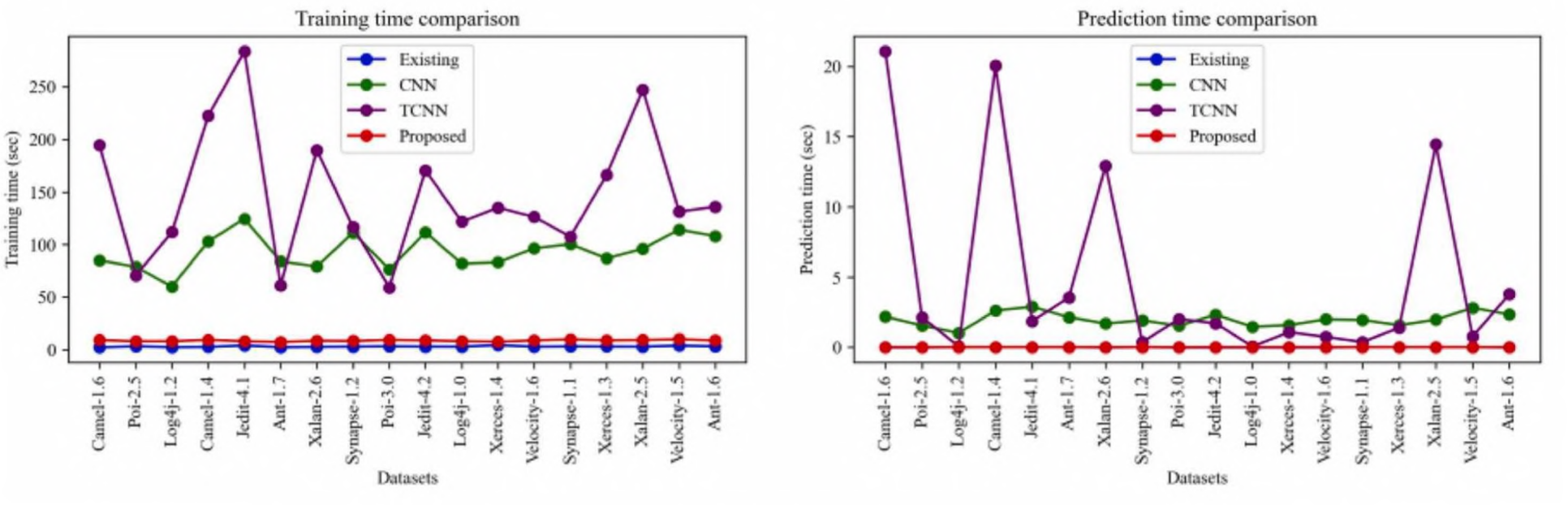

Рисунок В.1 - Порівняння складності моделей: тренування та прогнозування

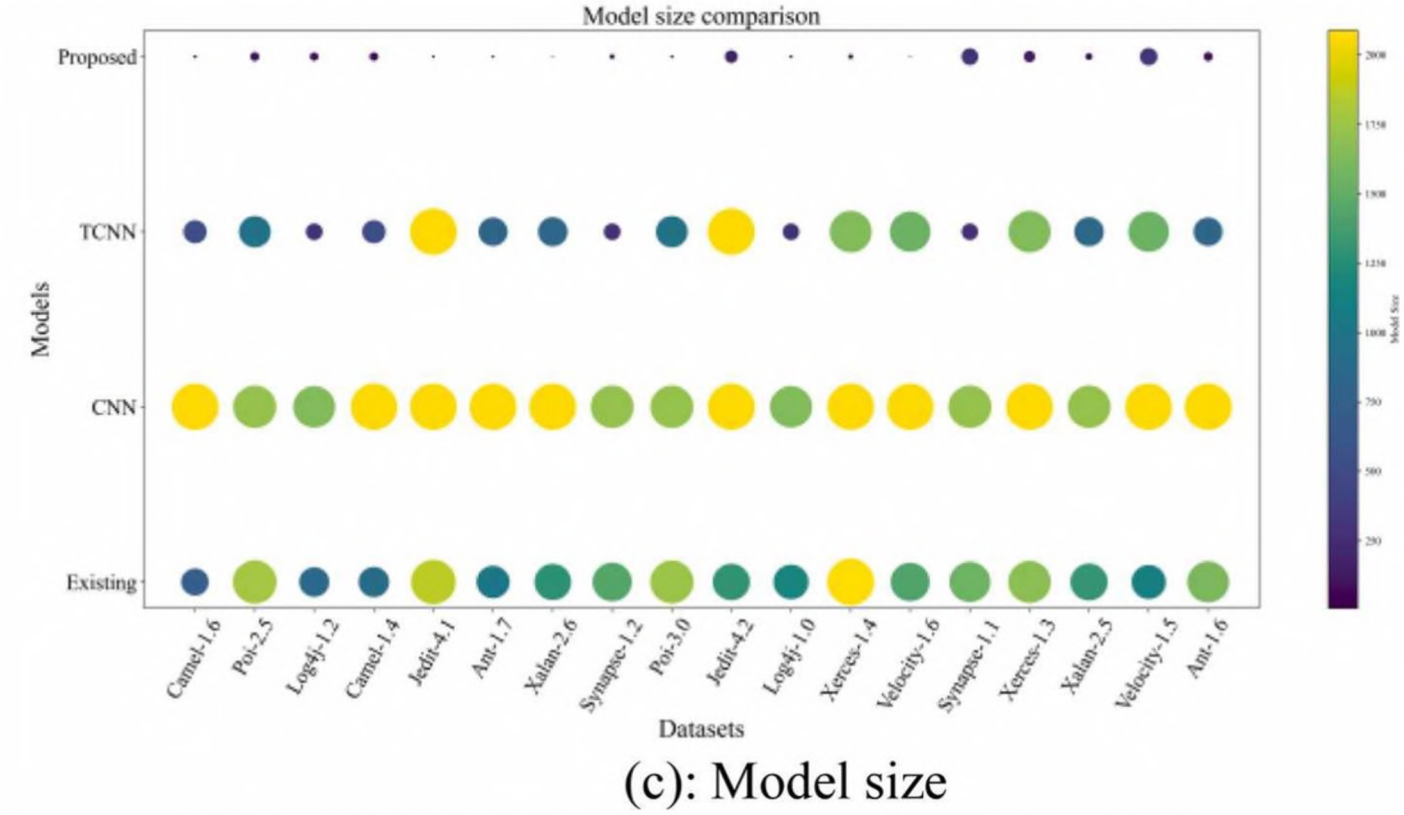

Рисунок В.2 - Порівняння складності моделей: моделювання

Додаток Г

Порівнянн метрик точності

Таблиця Г.1 - AUC для вибраних проєктів на основі існуючих метрик коду

Набори даних	DT	LR	SVM	KNN	MLP	RF	ADABOOST	XGBOOST
Camel−1.6	0.590	0.588	0.533	0.600	0.601	0.639	0.610	0.626
Poi−2.5	0.751	0.780	0.762	0.619	0.803	0.734	0.761	0.748
Log4j−1.2	0.706	0.824	0.773	0.739	0.738	0.770	0.677	0.762
Camel−1.4	0.574	0.679	0.589	0.661	0.668	0.691	0.670	0.678
Jedit−4.1	0.695	0.559	0.737	0.745	0.583	0.755	0.667	0.753
Ant−1.7	0.765	0.798	0.751	0.783	0.794	0.778	0.757	0.806
Xalan−2.6	0.528	0.624	0.581	0.610	0.608	0.674	0.584	0.596
Synapse−1.2	0.716	0.782	0.718	0.689	0.785	0.731	0.762	0.747
Poi−3.0	0.775	0.785	0.710	0.569	0.813	0.694	0.784	0.794
Jedit−4.2	0.753	0.679	0.792	0.726	0.734	0.811	0.773	0.794
Log4j−1.0	0.750	0.703	0.828	0.724	0.726	0.765	0.728	0.829
Xerces−1.4	0.646	0.680	0.678	0.615	0.649	0.794	0.740	0.777
Velocity−1.6	0.580	0.655	0.617	0.608	0.689	0.670	0.627	0.585
Synapse−1.1	0.695	0.697	0.682	0.654	0.700	0.714	0.678	0.694
Xerces−1.3	0.754	0.668	0.687	0.596	0.625	0.628	0.730	0.737
Xalan−2.5	0.606	0.607	0.620	0.619	0.641	0.642	0.597	0.617
Velocity−1.5	0.636	0.641	0.689	0.662	0.667	0.698	0.551	0.625
Ant−1.6	0.792	0.773	0.808	0.801	0.821	0.813	0.764	0.824
Середнє	0.684	0.696	0.697	0.668	0.703	0.722	0.692	0.722

Таблиця Г.2 - Точність для вибраних проєктів на основі існуючих метрик коду

Набори даних	DT	LR	SVM	KNN	MLP	RF	ADABOOST	XGBOOST
Camel−1.6	0.811	0.804	0.805	0.808	0.806	0.803	0.790	0.803
Poi−2.5	0.706	0.657	0.732	0.670	0.732	0.670	0.714	0.690
Log4j−1.2	0.706	0.706	0.779	0.733	0.752	0.788	0.651	0.804
Camel−1.4	0.809	0.830	0.832	0.832	0.836	0.785	0.809	0.829
Jedit−4.1	0.785	0.769	0.775	0.714	0.756	0.772	0.778	0.801
Ant−1.7	0.777	0.804	0.813	0.809	0.809	0.800	0.798	0.800
Xalan−2.6	0.579	0.627	0.594	0.583	0.603	0.628	0.581	0.612
Synapse−1.2	0.687	0.738	0.718	0.703	0.742	0.703	0.722	0.707
Poi−3.0	0.733	0.527	0.662	0.561	0.658	0.678	0.733	0.744
Jedit−4.2	0.795	0.833	0.850	0.795	0.782	0.825	0.874	0.838
Log4j−1.0	0.688	0.548	0.792	0.748	0.533	0.748	0.718	0.651
Xerces−1.4	0.413	0.433	0.428	0.437	0.431	0.437	0.505	0.435
Velocity−1.6	0.524	0.637	0.598	0.497	0.598	0.554	0.620	0.502
Synapse−1.1	0.693	0.680	0.702	0.657	0.698	0.716	0.639	0.680
Xerces−1.3	0.715	0.543	0.613	0.518	0.507	0.474	0.549	0.545
Xalan−2.5	0.591	0.555	0.596	0.607	0.599	0.627	0.574	0.606
Velocity−1.5	0.439	0.471	0.476	0.462	0.504	0.537	0.359	0.509
Ant−1.6	0.772	0.777	0.786	0.760	0.806	0.792	0.720	0.766
Середнє	0.679	0.663	0.697	0.661	0.675	0.685	0.674	0.684

Таблиця Г.3 - AUC для вибраних проєктів на основі запропонованих структурних метрик коду

Набори даних	DT	LR	SVM	KNN	MLP	RF	ADABOOST	XGBOOST
Camel−1.6	0.603	0.649	0.628	0.663	0.680	0.737	0.729	0.730
Poi−2.5	0.679	0.620	0.561	0.812	0.646	0.726	0.751	0.783
Log4j−1.2	0.822	0.847	0.853	0.817	0.849	0.799	0.750	0.808
Camel−1.4	0.711	0.660	0.666	0.705	0.680	0.760	0.735	0.780
Jedit−4.1	0.747	0.839	0.698	0.764	0.840	0.805	0.722	0.792
Ant−1.7	0.713	0.799	0.785	0.780	0.805	0.723	0.745	0.766
Xalan−2.6	0.537	0.714	0.706	0.643	0.669	0.761	0.575	0.627
Synapse−1.2	0.749	0.714	0.593	0.687	0.741	0.688	0.717	0.768
Poi−3.0	0.660	0.700	0.625	0.735	0.700	0.716	0.757	0.700
Jedit−4.2	0.707	0.821	0.712	0.736	0.779	0.788	0.703	0.779
Log4j−1.0	0.712	0.855	0.833	0.802	0.870	0.770	0.770	0.849
Xerces−1.4	0.784	0.832	0.858	0.891	0.966	0.874	0.680	0.868
Velocity−1.6	0.688	0.682	0.725	0.699	0.658	0.704	0.646	0.682
Synapse−1.1	0.689	0.732	0.615	0.703	0.663	0.696	0.721	0.683
Xerces−1.3	0.863	0.910	0.787	0.805	0.891	0.859	0.800	0.879
Xalan−2.5	0.664	0.642	0.669	0.688	0.683	0.694	0.649	0.721
Velocity−1.5	0.755	0.660	0.716	0.733	0.771	0.801	0.783	0.801
Ant−1.6	0.781	0.849	0.623	0.731	0.738	0.754	0.769	0.785
Середнє	0.715	0.751	0.703	0.744	0.757	0.759	0.722	0.767

Таблиця Г.4 - Точність для вибраних проєктів на основі запропонованих структурних метрик коду

Набори даних	DT	LR	SVM	KNN	MLP	RF	ADABOOST	XGBOOST
Camel−1.6	0.711	0.690	0.734	0.737	0.782	0.759	0.731	0.753
Poi−2.5	0.376	0.389	0.379	0.703	0.397	0.628	0.683	0.659
Log4j−1.2	0.779	0.761	0.724	0.743	0.761	0.761	0.743	0.752
Camel−1.4	0.805	0.732	0.795	0.774	0.800	0.823	0.808	0.825
Jedit−4.1	0.778	0.791	0.769	0.759	0.804	0.833	0.740	0.788
Ant−1.7	0.732	0.818	0.818	0.797	0.820	0.740	0.794	0.770
Xalan−2.6	0.544	0.570	0.606	0.606	0.625	0.672	0.546	0.598
Synapse−1.2	0.679	0.718	0.671	0.656	0.691	0.722	0.660	0.742
Poi−3.0	0.583	0.418	0.404	0.626	0.418	0.604	0.585	0.563
Jedit−4.2	0.817	0.792	0.836	0.768	0.814	0.820	0.683	0.833
Log4j−1.0	0.814	0.822	0.814	0.807	0.837	0.703	0.807	0.829
Xerces−1.4	0.552	0.520	0.523	0.545	0.532	0.581	0.583	0.574
Velocity−1.6	0.637	0.698	0.694	0.698	0.676	0.628	0.663	0.602
Synapse−1.1	0.738	0.761	0.693	0.725	0.657	0.711	0.734	0.698
Xerces−1.3	0.434	0.719	0.735	0.710	0.750	0.717	0.498	0.724
Xalan−2.5	0.594	0.570	0.591	0.612	0.605	0.638	0.601	0.636
Velocity−1.5	0.593	0.434	0.481	0.504	0.532	0.644	0.742	0.598
Ant−1.6	0.783	0.789	0.735	0.720	0.726	0.749	0.763	0.794
Середнє	0.664	0.666	0.667	0.694	0.679	0.707	0.687	0.708

Таблиця Г.5 - Порівняння інших метрик продуктивності

Набір даних	Помилковий рівень пропуску (Existing)	Помилковий рівень пропуску (Proposed)	Точність (Existing)	Точність (Proposed)	Повнота (Existing)	Повнота (Proposed)	F1-міра (Existing)	F1-міра (Proposed)
Camel−1.6	0.515	0.620	0.825	0.856	0.957	0.833	0.886	0.844
Poi−2.5	0.255	0.145	0.536	0.513	0.540	0.824	0.538	0.633
Log4j−1.2	0.281	0.361	0.818	0.808	0.875	0.819	0.845	0.813
Camel−1.4	0.681	0.528	0.857	0.884	0.891	0.909	0.873	0.896
Jedit−4.1	0.394	0.355	0.795	0.812	0.935	0.931	0.859	0.868
Ant−1.7	0.417	0.513	0.834	0.862	0.925	0.837	0.877	0.850
Xalan−2.6	0.401	0.158	0.654	0.574	0.647	0.972	0.651	0.722
Synapse−1.2	0.400	0.307	0.728	0.754	0.882	0.905	0.797	0.823
Poi−3.0	0.201	0.206	0.545	0.445	0.708	0.807	0.616	0.573
Jedit−4.2	0.590	0.614	0.956	0.916	0.836	0.890	0.892	0.903
Log4j−1.0	0.500	0.296	0.876	0.861	0.772	0.920	0.821	0.889
Xerces−1.4	0.026	0.000	0.310	0.376	0.980	1.000	0.472	0.547
Velocity−1.6	0.577	0.559	0.826	0.750	0.410	0.596	0.548	0.664
Synapse−1.1	0.520	0.557	0.836	0.795	0.759	0.790	0.796	0.792
Xerces−1.3	0.796	0.655	0.934	0.974	0.408	0.692	0.568	0.809
Xalan−2.5	0.369	0.274	0.625	0.604	0.701	0.860	0.661	0.710
Velocity−1.5	0.158	0.075	0.410	0.452	0.861	0.930	0.556	0.609
Ant−1.6	0.393	0.272	0.854	0.804	0.864	0.953	0.859	0.872
Середнє	0.415	0.361	0.734	0.724	0.775	0.859	0.729	0.768

Таблиця Г.6 - Порівняння з існуючими дослідженнями на основі показника AUC

Проєкти	CODEP	TCA+	HYDRA	TPTL	TDS	HIEL	CNN	TCNN	Запропонована
Camel−1.6	0.530	0.586	0.608	0.598	0.603	0.609	0.585	0.552	0.730
Poi−2.5	0.542	0.666	0.550	0.631	0.639	0.595	0.686	0.565	0.784
Log4j−1.2	0.542	0.532	0.785	0.690	0.412	0.500	0.686	0.596	0.808
Camel−1.4	0.538	0.641	0.884	0.587	0.536	0.919	0.609	0.583	0.781
Jedit−4.1	0.663	0.700	0.512	0.552	0.581	0.560	0.644	0.591	0.792
Ant−1.7	0.665	0.696	0.722	0.589	0.585	0.566	0.671	0.680	0.766
Xalan−2.6	0.578	0.547	0.599	0.633	0.574	0.671	0.621	0.620	0.628
Synapse−1.2	0.578	0.642	0.594	0.620	0.531	0.692	0.575	0.648	0.768
Poi−3.0	0.592	0.775	0.788	0.624	0.649	0.612	0.616	0.575	0.700
Jedit−4.2	0.725	0.689	0.500	0.587	0.538	0.537	0.618	0.634	0.779
Log4j−1.0	0.548	0.664	0.671	0.634	0.550	0.891	0.685	0.494	0.849
Xerces−1.4	0.579	0.602	0.615	0.597	0.409	0.578	0.710	0.557	0.868
Velocity−1.6	0.575	0.616	0.584	0.684	0.395	0.687	0.652	0.604	0.682
Synapse−1.1	0.611	0.595	0.656	0.522	0.563	0.767	0.602	0.632	0.684
Xerces−1.3	0.652	0.634	0.570	0.537	0.354	0.544	0.565	0.500	0.879
Xalan−2.5	0.565	0.564	0.696	0.650	0.515	0.730	0.582	0.583	0.722
Ant−1.6	0.672	0.697	0.714	0.584	0.623	0.592	0.660	0.662	0.786
Середнє	0.597	0.638	0.650	0.607	0.533	0.650	0.633	0.592	0.765

Таблиця Г.6 - Порівняння з існуючими дослідженнями на основі показника F1-score

Проєкти	CODEP	TCA+	HYDRA	TPTL	TDS	HIEL	CNN	TCNN	Запропонована
Camel−1.6	0.196	0.351	0.991	0.356	0.243	0.888	0.836	0.876	0.845
Poi−2.5	0.283	0.772	0.780	0.728	0.726	0.544	0.608	0.429	0.633
Log4j−1.2	0.256	0.652	0.914	0.606	0.538	0.143	0.169	0.125	0.814
Camel−1.4	0.208	0.375	0.503	0.339	0.274	0.912	0.857	0.897	0.897
Jedit−4.1	0.496	0.533	0.551	0.522	0.420	0.796	0.821	0.851	0.868
Ant−1.7	0.480	0.489	0.295	0.455	0.375	0.848	0.839	0.841	0.850
Xalan−2.6	0.336	0.543	0.593	0.533	0.394	0.689	0.647	0.655	0.722
Synapse−1.2	0.327	0.565	0.529	0.571	0.466	0.810	0.765	0.808	0.823
Poi−3.0	0.371	0.830	0.807	0.787	0.715	0.553	0.561	0.400	0.574
Jedit−4.2	0.429	0.332	0.492	0.370	0.217	0.840	0.836	0.866	0.903
Log4j−1.0	0.227	0.500	0.413	0.637	0.435	0.879	0.695	0.088	0.890
Xerces−1.4	0.301	0.493	0.903	0.690	0.744	0.816	0.009	0.006	0.547
Velocity−1.6	0.333	0.511	0.503	0.568	0.358	0.810	0.613	0.456	0.664
Synapse−1.1	0.385	0.458	0.494	0.475	0.440	0.861	0.779	0.776	0.793
Xerces−1.3	0.402	0.354	0.417	0.377	0.229	0.891	0.617	0.100	0.810
Xalan−2.5	0.336	0.543	0.593	0.533	0.394	0.689	0.595	0.618	0.710
Ant−1.6	0.516	0.541	0.807	0.595	0.415	0.831	0.830	0.859	0.873
Середнє	0.346	0.520	0.622	0.537	0.434	0.753	0.643	0.569	0.777

Додаток Д

Порівняння складності запропонованої моделі з існуючими моделями

Таблиця Д.1 - Порівняння складності запропонованої моделі з існуючими моделями

Датасети	Час навчання				Час прогнозування				Розмір моделі			
	Existing	CNN	TCNN	Proposed	Existing	CNN	TCNN	Proposed	Existing	CNN	TCNN	Proposed
Camel−1.6	2.13 s	85.07 s	194.73 s	9.18 s	0 s	2.19 s	21.07 s	0.02 s	725.6 KB	2057.8KB	528.8KB	16.2KB
Poi−2.5	3.25 s	78.70 s	70.52 s	8.09 s	0 s	1.56 s	2.14 s	0.02 s	1779.1 KB	1719.4KB	969.4KB	91.0 KB
Log4j−1.2	2.33 s	60.01 s	111.89 s	8.09 s	0.02 s	1.03 s	0.08 s	0.02 s	860.2 KB	1641.5KB	279.5KB	81.6 KB
Camel−1.4	2.57 s	102.98 s	222.41 s	9.36 s	0.02 s	2.62 s	20.04 s	0.02 s	894.6 KB	2057.9KB	528.8KB	83.9 KB
Jedit−4.1	4.06 s	124.49 s	283.88 s	8.15 s	0.02 s	2.90 s	1.87 s	0.02 s	1868.5 KB	2057.8KB	2057.6KB	12.7 KB
Ant−1.7	2.28 s	83.94 s	60.99 s	7.39 s	0.02 s	2.15 s	3.55 s	0.02 s	1038.7 KB	2057.9KB	803.0KB	12.3 KB
Xalan−2.6	2.64 s	79.16 s	189.86 s	8.59 s	0 s	1.70 s	12.92 s	0.02 s	1280.2 KB	2057.7KB	834.3KB	7.2 KB
Synapse−1.2	2.91 s	111.37 s	116.46 s	8.34 s	0.02 s	1.93 s	0.38 s	0.02 s	1437.1 KB	1719.6KB	279.4KB	30.0 KB
Poi−3.0	3.16 s	76.46 s	59.23 s	9.33 s	0 s	1.54 s	2.02 s	0.01 s	1746.8 KB	1719.4KB	969.4KB	14.6 KB
Jedit−4.2	2.94 s	111.67 s	170.54 s	8.88 s	0 s	2.33 s	1.72 s	0.02 s	1294.8 KB	2057.9KB	2057.6KB	170.3 KB
Log4j−1.0	2.83 s	81.88 s	121.70 s	8.18 s	0 s	1.47 s	0.07 s	0.02 s	1177.0 KB	1641.4KB	279.5KB	15.2 KB
Xerces−1.4	4.39 s	83.13 s	135.15 s	7.69 s	0 s	1.59 s	1.09 s	0.02 s	2089.1 KB	2057.8KB	1641.3KB	29.3 KB
Velocity−1.6	2.80 s	96.50 s	126.26 s	8.88 s	0 s	2.00 s	0.75 s	0.02 s	1418.8 KB	2057.7KB	1541.0KB	7.4 KB
Synapse−1.1	3.09 s	100.43 s	107.51 s	9.91 s	0.02 s	1.96 s	0.39 s	0.02 s	1551.1 KB	1719.6KB	279.4KB	288.5 KB
Xerces−1.3	2.95 s	86.86 s	166.24 s	8.97 s	0.02 s	1.58 s	1.40 s	0.02 s	1676.9 KB	2057.7KB	1641.3KB	141.5 KB
Xalan−2.5	2.84 s	96.05 s	247.31 s	9.24 s	0.02 s	1.98 s	14.45 s	0.02 s	1316.3 KB	1719.4KB	834.3KB	57.1 KB
Velocity−1.5	3.90 s	114.20 s	131.37 s	10.04 s	0.02 s	2.82 s	0.78 s	0.02 s	1132.5 KB	2057.9KB	1541.0KB	303.9 KB
Ant−1.6	3.19 s	108.13 s	135.88 s	8.76 s	0 s	2.35 s	3.79 s	0.02 s	1608.8 KB	2057.8KB	803.0KB	95.3 KB
Середнє	3.01 s	93.39 s	147.32 s	8.72 s	0.01 s	1.98 s	4.91 s	0.02 s	1383.1KB	1917.6KB	992.7KB	81.0KB

Додаток Ж

Набір метрик

Таблиця Ж.1 - Набір метрик, використаних у репозиторії Promise

Метрика коду	Тип метрики коду	Опис
LCOM3	Згуртованість	Розширення метрики LCOM [29]. Вона призначена для визначення зв'язності методів класу на основі спільних полів.
CAM	-	Призначена для обчислення ступеня зв'язності методів класу на основі переданих параметрів.
LCOM	-	Призначена для виявлення відсутності згуртованості між методами класу, і вона обчислює кількість несхожих пар методів у класі [17].
LOC	Розмір	Призначена для визначення розміру класу шляхом підрахунку рядків виконуваного коду всередині класу.
NPM	-	Призначена для підрахунку публічних методів класу [17].
AMC	Складність	Називається середньою складністю методу і є середньою цикломатичною складністю публічних та приватних методів класу, за винятком віртуальних та успадкованих методів [38].
Avg_cc	-	Називається середньою цикломатичною складністю і є середньою цикломатичною складністю всіх методів класу.
Max_cc	-	Називається максимальною цикломатичною складністю і призначена для знаходження методу з максимальною цикломатичною складністю у класі.
WMC	-	Призначена для обчислення суми цикломатичних складностей методів класу [17].
DAM	Інкапсуляція	Називається метрикою доступу до даних і є відношенням кількості приватних полів до кількості публічних, приватних і захищених полів класу [29].
MOA	-	Називається мірою агрегації і підраховує поля, визначені користувачем у класі [29].
MFA	Успадкування	Називається мірою функціональної абстракції, яка є відношенням кількості успадкованих методів до загальної кількості методів [29].
NOC	-	Призначена для підрахунку безпосередніх дочірніх (похідних) класів базового класу [17].
DIT	-	Називається глибиною дерева успадкування і підраховує глибину класу від кореня дерева успадкування [17].
IC	Зчеплення	Називається зчепленням за успадкуванням, яке підраховує кількість безпосередніх батьків класу [38].
RFC	-	Призначена для підрахунку методів, які викликаються об’єктом класу після отримання повідомлення [17].
CE	-	Називається ефірним зчепленням і підраховує вихідні виклики функцій [39].
CBM	-	Називається зчепленням між методами і підраховує кількість перевизначених методів [38].
CBO	-	Називається зчепленням між об'єктами і підраховує всі вхідні та вихідні виклики функцій [17].
CA	-	Називається аферентним зчепленням і підраховує вхідні виклики функцій [39].

Таблиця Ж.2 - Важливість ознак на основі VARL і ROC

Категорія	Метрика коду	VARL / Місце	ROC / Місце	Категорія	Метрика коду	VARL / Місце	ROC / Місце
Складність	ETCC	0.696 / (1)	0.575 / (1)	Згуртованість	ELCOCC	0.632 / (1)	0.531 / (2)
WMC	0.679 / (2)	0.508 / (4)	LCOM	0.634 / (2)	0.547 / (1)		
NPM	0.677 / (3)	0.517 / (3)	LCOM3	0.539 / (3)	0.514 / (3)		
LOC	0.644 / (4)	0.505 / (4)	CAM	0.636 / (4)	0.506 / (4)		
AMC	0.587 / (5)	0.549 / (2)	Успадкування	EIMCR	0.661 / (1)	0.571 / (1)	
MAX_CC	0.526 / (6)	0.517 / (4)	IC	0.564 / (2)	0.552 / (2)		
AVG_CC	0.491 / (7)	0.517 / (4)	NOC	0.526 / (3)	0.526 / (4)		
Зчеплення	ECAC	0.684 / (1)	0.578 / (1)	DIT	0.513 / (4)	0.504 / (5)	
CBO	0.668 / (2)	0.526 / (5)	MFA	0.51 / (5)	0.543 / (3)		
RFC	0.655 / (3)	0.507 / (6)	Інші	EOMCR	0.606 / (1)	0.612 / (1)	
CA	0.617 / (4)	0.539 / (4)	COC	0.585 / (2)	0.523 / (4)		
CE	0.576 / (5)	0.545 / (3)	MOA	0.567 / (3)	0.564 / (3)		
CBM	0.576 / (6)	0.552 / (2)	DAM	0.530 / (4)	0.572 / (2)		

Додаток З

Апробація отриманих результатів

ЗБІРНИК НАУКОВИХ ПРАЦЬ

З МАТЕРІАЛАМИ V МІЖНАРОДНОЇ НАУКОВОЇ КОНФЕРЕНЦІЇ

11 ЖОВТНЯ 2024 РІК

М. ЛУЦЬК, УКРАЇНА

**«СТРАТЕГІЧНІ НАПРЯМКИ РОЗВИТКУ НАУКИ:
ФАКТОРИ ВПЛИВУ ТА ВЗАЄМОДІЇ»**

УДК 082:001
С 83

Організація, від імені якої випущено видання:
ГО «Міжнародний центр наукових досліджень»
Номер запису організації в Єдиному реєстрі громадських об'єднань: 149914).

Голова оргкомітету: Сотник С.Г.

Верстка: Білоус Т.В.
Дизайн: Бондаренко І.В.

Рекомендовано до видання Вченою Радою Інституту науково-технічної інтеграції та співпраці. Протокол № 57 від 10.10.2024 року.

*Конференцію зареєстровано Державною науковою установою у сфері управління Міністерства освіти і науки «Український інститут науково-технічної експертизи та інформації» в базі даних науково-технічних заходів України на поточний рік та бюлетені «План проведення наукових, науково-технічних заходів в Україні» (**Посвідчення № 351 від 12.06.2024**).*

*Збірник наукових праць з матеріалами конференції видано офіційно суб'єктом видавничої справи зі **Свідоцтвом ДК № 7860 від 22.06.2023***

Матеріали конференції знаходяться у відкритому доступі на умовах ліцензії Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

С 83 **Стратегічні напрямки розвитку науки: фактори впливу та взаємодії**: збірник наукових праць з матеріалами V Міжнародної наукової конференції, м. Луцьк, 11 жовтня, 2024 р. / Міжнародний центр наукових досліджень. — Вінниця: ТОВ «УКРЛОГОС Груп, 2024. — 254 с.

ISBN 978-617-8440-16-9
DOI 10.62731/mcnd-11.10.2024

Викладено матеріали учасників V Міжнародної наукової конференції «Стратегічні напрямки розвитку науки: фактори впливу та взаємодії», яка відбулася 11 жовтня 2024 року у місті Луцьк.

УДК 082:001

© Колектив учасників конференції, 2024
© ГО «Міжнародний центр наукових досліджень», 2024
© ТОВ «УКРЛОГОС Груп», 2024

ISBN 978-617-8440-16-9

СЕКЦІЯ XI.
ХІМІЯ, ХІМІЧНА ТА БІОІНЖЕНЕРІЯ

ЗАСТОСУВАННЯ НЕОРГАНІЧНИХ СПОЛУК ДЛЯ НАДАННЯ ВОЛОКНИСТИМ МАТЕРІАЛАМ БАКТЕРИЦИДНИХ ВЛАСТИВОСТЕЙ У МЕДИЧНИХ І ПРОФІЛАКТИЧНИХ ЦІЛЯХ
Качківський В.В., Попович Т.А. **130**

СЕКЦІЯ XII.
ЕЛЕКТРОНІКА ТА ТЕЛЕКОМУНІКАЦІЇ

ЗАСТОСУВАННЯ ТЕХНОЛОГІЇ БЕЗДРОТОВОЇ ЗАРЯДКИ ЗА ДОПОМОГОЮ РЕКТЕН ДЛЯ ЖИВЛЕННЯ СИСТЕМ РЕЗ
Сокіркаєв Д.В. **135**

СЕКЦІЯ XIII.
ЕКОЛОГІЯ ТА ТЕХНОЛОГІЇ ЗАХИСТУ НАВКОЛИШНЬОГО СЕРЕДОВИЩА

ІМПЛЕМЕНТАЦІЯ ЄВРОПЕЙСЬКОГО ДОСВІДУ ЕКОЛОГІЧНОГО РЕГУЛЮВАННЯ ВИКОРИСТАННЯ ТЕРИТОРІЙ ПРИРОДНО-ЗАПОВІДНОГО ФОНДУ
Голян В.М., Іванців В.В. **138**

СЕКЦІЯ XIV.
КОМП'ЮТЕРНА ТА ПРОГРАМНА ІНЖЕНЕРІЯ

HARDWARE SUPPORT OF INFORMATION SYSTEMS
Pavlyk H.V. **142**

СЕКЦІЯ XV.
ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СИСТЕМИ

ПОБУДОВА ГНУЧКИХ СТРАТЕГІЙ УПРАВЛІННЯ ЗМІНАМИ В ML-ПРОЄКТАХ
Науково-дослідна група:
Висоцький А.В., Левандівський Н.Б., Вівюрка Н.М., Сулима Б.Я. **151**

ПРИКЛАДНІ АСПЕКТИ ВИКОРИСТАННЯ МАШИННОГО НАВЧАННЯ ДЛЯ ОБРОБКИ ТЕКСТОВИХ ДАНИХ
Науково-дослідна група:
Колівушко Е., Лучка С., Кондратюк Г., Кравчук Б., Тарасюк С. **154**

СЕКЦІЯ XV.
ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СИСТЕМИ

ПОБУДОВА ГНУЧКИХ СТРАТЕГІЙ УПРАВЛІННЯ ЗМІНАМИ В ML-ПРОЄКТАХ

НАУКОВО-ДОСЛІДНА ГРУПА:

Висоцький Андрій Васильович
здобувач вищої освіти факультету комп'ютерних інформаційних технологій
Західноукраїнський національний університет, Україна

Левандівський Назарі Богданович
здобувач вищої освіти факультету комп'ютерних інформаційних технологій
Західноукраїнський національний університет, Україна

Вівюрка Назар Миколайович
здобувач вищої освіти факультету комп'ютерних інформаційних технологій
Західноукраїнський національний університет, Україна

Сулима Богдан Ярославович
здобувач вищої освіти факультету комп'ютерних інформаційних технологій
Західноукраїнський національний університет, Україна

Гнучке управління змінами (Agile change management) – це підхід до управління, що дозволяє організаціям швидко адаптуватися до змін у вимогах або умовах проєкту. В IT-проєктах цей підхід має вирішальне значення через постійний технологічний прогрес і непередбачувані зовнішні фактори. Ключовою перевагою гнучкого управління є його здатність забезпечувати швидку адаптацію до змін і зменшення ризиків при впровадженні нових технологій, таких як машинне навчання (ML). Проєкти, що використовують ML, часто вимагають регулярних оновлень алгоритмів і змін у даних, що робить необхідним постійне гнучке реагування на нові вимоги. Наприклад, дослідження показують, що у проєктах з ML гнучкі методи допомагають швидше інтегрувати нові моделі без значних затримок у розробці [1].

Традиційні підходи до управління змінами, такі як Waterfall або PRINCE2, передбачають послідовне виконання етапів проєкту з

фіксованими вимогами на початкових стадіях. Основна увага приділяється детальному плануванню та контролю, що робить ці підходи менш гнучкими при виникненні нових обставин або технологічних інновацій, таких як впровадження машинного навчання. Наприклад, у традиційних проєктах машинного навчання зміни в даних або моделях можуть спричиняти затримки та перевитрати бюджету через необхідність перепроектування на пізніх етапах [2].

У швидкозмінних середовищах, таких як проєкти з машинного навчання, традиційні підходи мають декілька значних обмежень:

- Низька адаптивність: оскільки вимоги до проєктів з ML часто змінюються на основі нових даних або вдосконалення моделей, традиційні підходи не дозволяють швидко реагувати на такі зміни.
- Довгі цикли розробки: проєкти з фіксованими етапами можуть не встигати за швидкими інноваціями в галузі ML, що призводить до затримок [3].
- Непередбачуваність результатів: традиційні методи можуть не забезпечити належної гнучкості для адаптації моделей ML, особливо коли результати вимагають змін у самій інфраструктурі або способі обробки даних [4].

Для підтримки гнучкого управління змінами в IT-проєктах, що використовують машинне навчання, існують різні інструменти, які дозволяють знизити ризики та забезпечити ефективну адаптацію. Серед найбільш популярних є:

- JIRA – для управління задачами та відстеження змін у вимогах проєкту. JIRA широко використовується в проєктах, де необхідна інтеграція з ML-процесами для відслідковування прогресу в моделюванні [5].
- TensorFlow Extended (TFX) – інструмент для моніторингу і відстеження якості моделей ML, що дозволяє швидко інтегрувати нові зміни в моделі та алгоритми [6].
- MLflow – платформа для керування життєвим циклом моделей машинного навчання, яка дозволяє відстежувати експерименти, керувати моделями і автоматизувати процеси оновлення моделей в рамках гнучкого підходу [7].

Гнучке управління змінами є критичним для успіху IT-проєктів, особливо у сфері машинного навчання, де швидке адаптування до нових даних і технологій є необхідністю. Проєкти, які використовують ML,

виграють від гнучких методологій завдяки можливості інтеграції нових моделей та алгоритмів без значних затримок у виконанні. Важливою рекомендацією є використання спеціалізованих інструментів для відстеження змін і оновлень у проєкті. Крім того, для підвищення ефективності необхідно впроваджувати Agile-практики, що забезпечують гнучкість на всіх етапах розробки та експлуатації.

.

Список використаних джерел:

1. Smith, J. (2021). Agile Change Management in Machine Learning Projects. *Journal of Software Development*, 23(4), 45-67.
2. Brown, P., Doe, A., & Harris, L. (2020). Challenges of Traditional Change Management in IT Projects. *IT Management Review*, 12(2), 98-115.
3. Johnson, K., & Walker, R. (2019). Managing Delays in Machine Learning Projects Using Traditional Methods. *Software Engineering Quarterly*, 33(1), 22-34.
4. Williams, M. (2020). Predicting the Success of Machine Learning Models in Dynamic Environments. *Data Science Journal*, 18(3), 90-110.
5. Atlassian (2022). JIRA Software. Retrieved from https://www.atlassian.com/software/jira
6. Abadi, M., Barham, P., Chen, J., et al. (2016). TensorFlow: A System for Large-Scale Machine Learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 265-283.
7. Zaharia, M., Chen, A., & Davidson, J. (2018). MLflow: A Platform for Managing the Machine Learning Lifecycle. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*, 1211-1219.

МАТЕРІАЛИ

VI ВСЕУКРАЇНСЬКОЇ СТУДЕНТСЬКОЇ НАУКОВОЇ КОНФЕРЕНЦІЇ

20 ВЕРЕСНЯ 2024 РІК • М. КРОПИВНИЦЬКИЙ, УКРАЇНА

РОЗВИТОК СУЧАСНОЇ НАУКИ: АКТУАЛЬНІ ПИТАННЯ ТЕОРІЇ ТА ПРАКТИКИ

ISBN 978-617-8440-05-3
DOI 10.36074/liga-ukr-20.09.2024

УДК 082:001
Р 64

Голова оргкомітету: Коренюк І.О.
Верстка: Зрада С.І.
Дизайн: Бондаренко І.В.

Рекомендовано до видання Вченою Радою Інституту науково-технічної інтеграції та співпраці. Протокол № 54 від 19.09.2024 року.

Конференцію зареєстровано Державною науковою установою «УкрІНТЕІ» в базі даних науково-технічних заходів України та інформаційному бюлетені «План проведення наукових, науково-технічних заходів в Україні» (Посвідчення №317 від 12.06.2024).

Матеріали конференції знаходяться у відкритому доступі на умовах ліцензії CC BY-SA 4.0 International.

Р 64 **Розвиток сучасної науки: актуальні питання теорії та практики**: матеріали VI Всеукраїнської студентської наукової конференції, м. Кропивницький, 20 вересня, 2024 рік / ГО «Молодіжна наукова ліга». — Вінниця: ТОВ «УКРЛОГОС Груп», 2024. — 212 с.

ISBN 978-617-8440-05-3
DOI 10.62732/liga-ukr-20.09.2024

Викладено матеріали учасників VI Всеукраїнської мультидисциплінарної студентської наукової конференції «Розвиток сучасної науки: актуальні питання теорії та практики», яка відбулася 20 вересня 2024 року у місті Кропивницький, Україна.

УДК 082:001

© Колектив учасників конференції, 2024
© ГО «Молодіжна наукова ліга», 2024
© ТОВ «УКРЛОГОС Груп», 2024

ISBN 978-617-8440-05-3

АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО УПРАВЛІННЯ УРАЗЛИВОСТЯМИ В БАНКІВСЬКИХ ЦИФРОВИХ СИСТЕМАХ
Сиопок С.М., ***Науковий керівник: Снігуров А.В.*** **..... 91**

МЕТОДИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ОБСЛУГОВУВАННЯ КЛІЄНТІВ БАНКУ
Крисак І., ***Науковий керівник: Литвиненко О.*** **..... 93**

ПЕРЕВАГИ МАШИННОГО НАВЧАННЯ В УПРАВЛІННІ ПРОЕКТАМИ
Висоцький А.В., ***Левандівський Н.Б., Вівюрка Н.М.*** **..... 95**

СЕКЦІЯ 13.
ФІЗИКО-МАТЕМАТИЧНІ НАУКИ

ДОСЛІДЖЕННЯ ЗМІНИ ПОКАЗНИКІВ КРОВІ МІКРОСКОПІЧНИМ МЕТОДОМ
Пальченко А.О. 97

ФРАКТАЛЬНА ГЕОМЕТРІЯ ТА ЇЇ ЗАСТОСУВАННЯ У МОДЕЛЮВАННІ ПРИРОДНИХ ОБ'ЄКТІВ ТА ЯВИЩ
Косяк А.В., ***Науковий керівник: Ромащук Л.В.*** **..... 101**

СЕКЦІЯ 14.
ФІЛОЛОГІЯ ТА ЖУРНАЛІСТИКА

ЗАСОБИ СТВОРЕННЯ САТИРИ В РОМАНІ МО ЯНЯ "КРАЇНА ВИНА"
Долженко А.О., ***Науковий керівник: Жукова К.Є.*** **..... 104**

КОНФУЦІАНСЬКІ МОТИВИ У ДЗЕРКАЛІ КИТАЙСЬКОЇ ФРАЗЕОЛОГІЇ
Вілісова А.В., ***Науковий керівник: Руда Н.В.*** **..... 107**

ЛІНГВОПОЕТИКА РОМАНУ "ДРАКУЛА" БРЕМА СТОКЕРА: МЕТАФОРА ВАМПІРИЗМУ
Чернега А., ***Науковий керівник: Коробкова Н.*** **..... 109**

СЕМАНТИКО-ПРАГМАТИЧНИЙ ПОТЕНЦІАЛ МОЛОДІЖНОГО СЛЕНГУ В УКРАЇНСЬКІЙ МОВІ
Зданевич О.М., ***Науковий керівник: Коваль Л.М.*** **..... 111**

СУФІКСАЛЬНИЙ СПОСІБ ТВОРЕННЯ АНГЛІЙСЬКИХ ТЕРМІНІВ АВІАЦІЙНОГО СПРЯМУВАННЯ
Жученко Є.О., ***Науковий керівник: Гелетка М.Л.*** **..... 113**

СЕКЦІЯ 15.
ПЕДАГОГІКА ТА ОСВІТА

ВИЯВЛЕННЯ СТАНУ АДАПТОВАНОСТІ ДІТЕЙ З ОСОБЛИВИМИ ОСВІТНІМИ ПОТРЕБАМИ ДО УМОВ ЗАКЛАДУ ЗАГАЛЬНОЇ СЕРЕДНЬОЇ ОСВІТИ
Тичина С.В., ***Науковий керівник: Хлєбік С.Р.*** **..... 117**

Висоцький Андрій Васильович, здобувач вищої освіти
факультету комп'ютерних інформаційних технологій
Західноукраїнський національний університет, Україна

Левандівський Назарій Богданович, здобувач вищої освіти
факультету комп'ютерних інформаційних технологій
Західноукраїнський національний університет, Україна

Вівюрка Назар Миколайович, здобувач вищої освіти
факультету комп'ютерних інформаційних технологій
Західноукраїнський національний університет, Україна

ПЕРЕВАГИ МАШИННОГО НАВЧАННЯ В УПРАВЛІННІ ПРОЕКТАМИ

Машинне навчання набуває все більшого значення в управлінні проектами, допомагаючи підвищувати ефективність, покращувати прогнозування та оптимізувати ресурси. Метою цього огляду є аналіз поточного стану досліджень у цій сфері, представлений у п'яти (Таблиця 1) обраних наукових публікаціях.

Таблиця 1

Переваги машинного навчання в управлінні проектами

Публікація	Переваги машинного навчання в управлінні проектами
Роль штучного інтелекту в управлінні проєктами (В. Баранов, 2021)	Оптимізація ресурсів, підвищення ефективності управління, прогнозування ризиків.
Штучний інтелект в управлінні проєктами (Д.В. Хоронжук, 2023)	Автоматизація процесів управління, покращення прогнозування та оптимізація ресурсів.
Еволюція управління проектами в транспортній логістиці (В. Бабій, А. Михалків, Ю. Розумний, 2024)	Гнучкість у реагуванні на зміни, покращення управління транспортними логістичними проектами.
Застосування методів машинного навчання в управлінні іт проектами (О.О. Гудь, Н.Е. Кунанець, 2024)	Підвищення ефективності управління, вирішення проблем в IT-проєктах, аналіз великих обсягів даних.
Сучасні інформаційні технології в системі управління проектами (В. Баранов, 2021)	Використання інноваційних технологій, поліпшення управління різними типами проектів.

Аналіз представленої таблиці показує, що машинне навчання має низку значних переваг в управлінні проектами. У публікації В. Баранова (2021) підкреслюється важливість машинного навчання для оптимізації ресурсів, підвищення ефективності управління та прогнозування ризиків. Д.В. Хоронжук (2023) зазначає, що машинне навчання сприяє автоматизації процесів управління, покращенню прогнозування та оптимізації ресурсів. Робота В. Бабія, А. Михалківа та Ю. Розумного (2024) акцентує увагу на гнучкості у реагуванні на зміни та покращенні управління в транспортній логістиці. О.О. Гудь і Н.Е. Кунанець (2024) висвітлюють ефективність управління, вирішення проблем в IT-проєктах та аналіз великих обсягів даних за допомогою методів машинного навчання. Нарешті, друга публікація В. Баранова (2021) підкреслює роль інноваційних технологій у покращенні управління різними типами проектів. Загалом, ці публікації свідчать про значний потенціал машинного навчання у різних аспектах управління проектами, сприяючи підвищенню їх ефективності,

гнучкості та результативності.

Огляд показав, що машинне навчання стає невід'ємною частиною управління проектами. Різні аспекти МН, такі як прогнозування ризиків, оптимізація ресурсів та підвищення ефективності, досліджуються у ряді наукових публікацій. Впровадження МН у різні сфери управління проектами демонструє значний потенціал для вдосконалення сучасних методів управління.

Список використаних джерел:

1. В. Баранов. Роль штучного інтелекту в управлінні проектами. 2021. http://umo.edu.ua/images/content/institutes/imp/struktura/kaf_upravl_proekt/material/%D0%9C%D0%9E%D0%9D%D0%9E%D0%93%D0%A0%D0%90%D0%A4%D0%98%D0%AF_%D0%9A%D0%B0%D1%80%D1%82%D0%B0%D1%88%D0%BE%D0%B2_%D0%94%D1%83%D0%B1%D0%B8%D0%BD%D0%B8%D0%BD%D0%B0_2021_%D0%B2%D0%B5%D1%80%D1%81%D1%82%D0%BA%D0%B0%20(1).pdf#page=302.
2. Д.В. Хоронжук. Штучний інтелект в управлінні проектами. 2023. https://www.mnau.edu.ua/files/nauk_prof_konf/zbirnyk-tez-17-03-23.pdf#page=126.
3. В. Бабій, А. Михалків, Ю. Розумний. Еволюція управління проектами в транспортній логістиці: виклики та інноваційні рішення. 2024. https://elartu.tntu.edu.ua/bitstream/lib/44877/2/VII_MCNTK_2024_Babii_V-Evolution_of_project_management_199-200.pdf.
4. О.О. Гудь, Н.Е. Кунанець. Застосування методів машинного навчання в управлінні іт проектами. 2024. https://isu-conference.com/wp-content/uploads/2024/04/Innovations_in_scientific_research_world_experience_and_realities_April_10_12_2024_Riga_Latvia.pdf#page=92.
5. В. Баранов. Сучасні інформаційні технології в системі управління проектами. 2021. https://econom.lnu.edu.ua/wp-content/uploads/2016/09/Zbirnyk-Kropyvnytskyy-17.12.2021.pdf#page=26.