Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Метод інтелектуального аналізу комп'ютерних систем для спеціалізованих гетерогенних обчислювальних платформ
(тема)

Виконав:
здобувач 2 року навчання,
групи СПм-23-3
Дмитро ІРХА
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма
Системне програмування
(повна назва освітньої програми)

Керівник: проф. Георгій КУЧУК
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Ірхі Дмитру Максимовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Метод інтелектуального аналізу комп'ютерних систем для спеціалізованих гетерогенних обчислювальних платформ

затверджена наказом по університету від “ 21 ” квітня 2025 р. № 296 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 16 червня 2025 р.

3. Вхідні дані до роботи Python, навчальна вибірка, KNN, SVM, RF, ELM

4. Перелік питань, що потрібно опрацювати у роботі

1) Дослідження нтелектуального аналізу програмних систем.
2) Методи і алгоритми винесення рекомендацій по прискорення програм
3) Методи і алгоритми обліку навчальних досягнень в курсах по програмування
4) Висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 14 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Огляд літературних джерел та аналіз	22.04.25-29.04.25	
	предметної області досліджень		
2	Вибір та обгрунтування методики дослідження	30.04.25-05.05.25	
3	Вибір ШНМтрументальних засобів	06.05.25-09.05.25	
4	Розробка моделей протоколів	10.05.25-21.05.25	
5	Проведення експериментів	22.05.25-02.06.25	
6	Оформлення матеріалів кваліфікаційної роботи	03.06.25-05.06.25	
7	Подання кваліфікаційної роботи керівникові та	06.06.25-09.06.25	
	її попередній захист		
8	Подання кваліфікаційної роботи на	10.06.25-12.06.25	
	рецензування		

Дата видачі завдання " 21 " квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Георгій КУЧУК
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 77 с., 26 рисунок, 16 табл., 1 дод., 14 джерел.

КОМП’ЮТЕРНА СИСТЕМА, ЦИФРОВИЙ АССИСТЕНТ ВИКЛАДАЧА, ШТУЧНА НЕЙРОННА МЕРЕЖА

Метою кваліфікаційної роботи є підвищення продуктивності виконання програм для спеціалізованих гетерогенних обчислювальних платформ Об'єктом дослідження є процес інтелектуального аналізу комп'ютерних систем для спеціалізованих гетерогенних обчислювальних платформ. Предмет дослідження програмні системи для спеціалізованих гетерогенних обчислювальних платформ.

В ході виконання кваліфікаційної роботи було проведено аналіз сучасних підходів до проектування програмних і апаратних систем, для інтелектуального аналізу текстів програм. Розроблено підход до вилучення ознак текстів програм. Проведено експериментальне дослідження методу інтелектуального аналізу, для прискорення виконання на гетерогенній обчислювальній платформі.

ABSTRACT

Master’s thesis: 77 pages, 26 figures, 16 tables, 1 appendices, 14 sources.

COMPUTER SYSTEM, DIGITAL TEACHER ASSISTANT, ARTIFICIAL NEURAL NETWORK

The purpose of the qualification work is to increase the productivity of program execution for specialized heterogeneous computing platforms. The object of the study is the process of intelligent analysis of computer systems for specialized heterogeneous computing platforms. The subject of the study is software systems for specialized heterogeneous computing platforms.

During the qualification work, an analysis of modern approaches to the design of software and hardware systems was conducted for intelligent analysis of program texts. An approach to extracting features of program texts was developed. An experimental study of the method of intelligent analysis was conducted to accelerate execution on a heterogeneous computing platform.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 8
ВСТУП 9
1 ДОСЛІДЖЕННЯ НТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ПРОГРАМНИХ СИСТЕМ 11
1.1 Програмні системи для гетерогенних обчислювальних платформ 11
1.3 Визначення завдання по тексту програми 22
2 МЕТОДИ І АЛГОРИТМИ ВИНЕСЕННЯ РЕКОМЕНДАЦІЙ ПО ПРИСКОРЕННЯ ПРОГРАМ 27
2.1 Винесення рекомендацій по підвищення швидкості виконання програм 27
2.2 Методи і алгоритми пошуку фрагментів текстів програм по програмі- прикладу 29
2.3 Оцінка якості векторних уявлень текстів програм на основі ланцюгів Маркова 33
2.4 Алгоритм пошуку фрагментів текстів програм по програмі-прикладу 36
2.5 Результати застосування методики винесення рекомендацій по прискорення програм 42
3 МЕТОДИ І АЛГОРИТМИ ОБЛІКУ НАВЧАЛЬНИХ ДОСЯГНЕНЬ В КУРСАХ ПО ПРОГРАМУВАННЯ 48
3.1 Методика обліку навчальних досягнень в масових курсах по програмування 48
3.2 Виявлення способів рішення завдань на основі кластеризації програм 52
3.3 Визначення способів рішення завдань на основі класифікації програм 61

3.4.Автоматичний облік навчальних досягнень 62
ВИСНОВОК..... 67
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ..... 68
ДОДАТОК А Графічний матеріал кваліфікаційної роботи..... 70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПС – програмна система

КС – комп’ютерна система

ЦАВ – цифровий ассистент викладача

ШНМ – штучна нейронна мережа

CPU – central processing unit

GPU – graphics processing unit

AST – abstract syntax tree, дерево абстрактного синтаксису

ASIC – application-specific integrated circuit

FPGA – field-programmable gate array

OpenCL – open computing language

CUDA – compute unified device architecture

HDL – hardware description language

VHDL – very high-speed integrated circuits hardware description language

SQL – structured query language

GA – genetic algorithm

ER – entity relationship

LMS – learning management system

RBF – radial-basis function

SIMD – single instruction, multiple data

ВСТУП

Комп'ютеризація і цифровізація наводить до розширення кола завдань, що стоять перед розробниками програмних систем (ПС). Для прискорення і спрощення розробки ПС були запропоновані методи інтелектуального статичного аналізу текстів програм для задач пошуку помилок, уразливостей, порушень стандартів програмування, а також для задач відновлення назв ідентифікаторів при зворотній розробці ПС, визначення функціональних можливостей та авторства ПС.

Методи інтелектуального аналізу засновані на перетворенні текстів програм в векторні уявлення з наступним застосуванням методів та алгоритмів кластеризації та класифікації, при цьому існує потреба у підвищенні якості аналізу текстів програм. Інновації у архітектурі серверів дозволяють створювати гетерогенні обчислювальні платформи. У гетерогенних обчислювальних платформах центральний процесор доповнюється спеціалізованими прискорювачами, такими як: графічні процесори, інтегральні схеми спеціального призначення, програмовані логічні інтегральні схеми.

Завдання апаратно-програмного розбиття в даний час вирішується на основі виявлених вручну розробником фрагментів текстів програм, що володіють потенціалом прискорення на заданої гетерогенної обчислювальної платформи, та внаслідок цього існує потреба в прискорення ПС на основі можливостей, що надаються гетерогенною обчислювальною платформою із заданим набором апаратних прискорювачів, на якої планується розгортання КС, для досягнення найкращої швидкості роботи КС. У зв'язку з цим актуальна розробка математичного та алгоритмічного забезпечення інтелектуального статичного аналізу текстів програм ПС з метою їх перетворення для підвищення швидкості виконання програм на заданій гетерогенній обчислювальній платформі.

Об'єктом дослідження є процес інтелектуального аналізу комп'ютерних

систем для спеціалізованих гетерогенних обчислювальних платформ.

Предмет програмні системи для спеціалізованих гетерогенних обчислювальних платформ.

Метою роботи підвищення продуктивності виконання програм для спеціалізованих гетерогенних обчислювальних платформ.

Для досягнення мети роботи необхідно вирішити наступні завдання:

Провести аналізу сучасних підходів до проектування програмних і апаратних систем, для інтелектуального аналізу текстів програм.

Розробити підход до вилучення ознак текстів програм.

Провести експериментальне дослідження методу інтелектуального аналізу, для прискорення виконання на гетерогенній обчислювальній платформі.

Методи досліджень включають методи теорії формальних мов, теорії графів, теорії оптимізації, теорії інтелектуальних систем, теорії ймовірностей та математичної статистики, системного аналізу. У процесі проведення експериментальних досліджень застосовувалися методи математичного та імітаційного моделювання.

Наукова новизна полягає в тому, що удосконалено метод перетворення текстів програм на векторні уявлення, заснований на побудові ланцюгів Маркова.

1 ДОСЛІДЖЕННЯ НТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ПРОГРАМНИХ СИСТЕМ

1.1 Програмні системи для гетерогенних обчислювальних платформ

Інновації в архітектурі обчислювальних машин забезпечили можливість створення гетерогенних обчислювальних платформ для рішення спеціалізованих завдань [1]. У гетерогенних платформах центральний процесор загального призначення доповнюється спеціалізованими прискорювачами, такими як графічні процесори [2], інтегральними схемами спеціального призначення, программними логічними інтегральними схемами [3]. Сучасні тенденції в області проектування ПС спрямовані на перенесення програмних реалізацій обчислювально-інтенсивних алгоритмів [4-6] на апаратні прискорювачі, доступні на гетерогенної обчислювальної платформі, для підвищення швидкості виконання програм за рахунок балансування навантаження, запобігання простою процесорів [7]. Вибір спеціалізованого процесора для апаратного прискорення фрагменту тексту програми залежить від алгоритму, що реалізується фрагментом. Для максимізації продуктивності програмно-апаратних комплексів з урахуванням заданих обмежень, таких як, наприклад, обмеження на енергоспоживання, широке розповсюдження отримав підхід codesign [2, 3], згідно з яким проектування програмних та апаратних коштів виконується спільно з метою залучення спеціалізованих прискорювачів під час вирішення обчислювальних завдань у процесі функціонування ПС.

При розробці ПС, як правило, використовується один або кілька мов програмування загального призначення, таких як Python, Java, C#, C, JavaScript, тоді як програмування апаратних прискорювачів застосовують спеціалізовані мови. Наприклад, Open Computing Language (OpenCL) [14] і Compute Unified Device Архітектура (CUDA) [15] використовуються для

програмування графічних процесорів, а для конфігурування FPGA традиційним способом застосовуються мови опису апаратури (англ. Hardware Description Language, HDL), Verilog [6] або VHDL. Для спрощення перенесення алгоритмів на FPGA розроблено інструменти високорівневого синтезу інструкції рівня регістрових передач [8]. Такі інструменти дозволяють транслювати алгоритми, описані на мовами високого рівня, таких як C, C++, OpenCL, RTL-код для подальшого конфігурування FPGA.

Відомо також про предметно-орієнтовані мови програмування, що дозволяють синтезувати високопродуктивний HLS код [2].

Для постановки і рішення завдання апаратно-програмного розбиття необхідно попередньо виділити фрагменти текстів програм ПС, які мають потенціал апаратного прискорення, а також отримати версії виділених фрагментів, придатні для запуску на апаратних прискорювачів. У цією зв'язку сучасні дослідження в області статичного аналізу присвячені ідентифікації фрагментів ПС з метою їхньої подальшої переробки для перенесення на спеціалізовані апаратні прискорювачі [6, 9].

Для підвищення продуктивності ПС, функціонуючої на гетерогенної обчислювальній платформі, реалізованій з використанням мов програмування загального призначення, необхідно ідентифікувати фрагменти текстів програм, придатні для прискорення, а також вибрати відповідний спосіб їх прискорення. Способи прискорення фрагментів текстів програм ПС можуть включати як задіяння спеціалізованих процесорів для запобігання їх простою, так і використання паралелізму за даними багатоядерного процесора загального призначення [2, 3]. І в тому, і в другом випадку для спрощення і прискорення розробки може бути використаний вбудований DSL, заснований на синтаксисі мови програмування спільного призначення, підтримуючий трансляцію в придатне для запуску на спеціалізованому процесорі уявлення. За наявності доступу до текстів програм ПС завдання ідентифікації фрагментів текстів програм ПС для винесення рекомендацій по їх прискорення може бути вирішена методами статичного аналізу. Перші методи статичного аналізу та

перетворення програм були описані в [3].

Завдання ідентифікації фрагментів текстів програм ПС для винесення рекомендацій по підвищенню їх швидкості виконання може бути зведено до задачі формування бази програм-прикладів і варіантів підвищення їх швидкості виконання при допомоги прискорювачів, доступних на обчислювальної платформі, а також до задачі пошуку фрагментів текстів програм ПС по програмі-прикладу. Пошук фрагментів за програмою-прикладом у цьому випадку виконується для виявлення фрагментів, реалізують або той ж алгоритм, що і шукана програма-приклад, або інший алгоритм, але що володіє схожими з алгоритмом програми-прикладу властивостями.

Одне з перших згадок завдання визначення втіленого в тексті програми алгоритму зустрічається при опису метода, заснованого на отриманні та порівнянні чисельних характеристик текстів програм. Такі чисельні характеристики включають, наприклад, цикломатичну складність [6], число змінних і операторів присвоєння, число унікальних операторів і операндів. Завдання рекомендації назви для програмного методу або Функція може бути зведена до завдання визначення алгоритму, що реалізується аналізованим методом або функцією.

1.2. Аналіз перетворення текстів програм у векторні уявлення

У задачі визначення реалізованого алгоритму при побудові відображення g: X → Rm, що виконує витяг ознак текстів програм з X для формування їх m-вимірних векторних уявлень, необхідно враховувати, що деталі реалізації одного і того ж алгоритму можуть незначно відрізнятися в різних програмах. алгоритми повинні бути віддалені один від одного, а векторні уявлення текстів програм, що реалізують схожі алгоритми, повинні розташовуватися близько один до одного.

Перетворення текстів програм у вектори на основі AST (рисунок 1.1)

дозволяє врахувати ієрархічні зв'язки між синтаксичними елементами [9], що вигідно відрізняє даний підхід від підходів на основі аналізу ключових слів текстів програм [5] 2, не враховують ієрархічні зв'язки між елементами синтаксису, а такі підходи, як code2vec [9], засновані на вилученні шляхів з AST та навчанні ШНМ, складні в реалізації, характеризуються високою обчислювальною складністю і можуть не дозволити досягти високої точності в ряді практичних завдань [7].

Пропонований підхід до перетворення текстів програм у векторні уявлення на основі ланцюгів Маркова порядку n, побудова яких виробляється для AST [6], дозволяє, з одного боку, врахувати ієрархічні зв'язки між елементами AST, а, з іншого боку, характеризується низькою обчислювальною навчанні ШНМ [9].

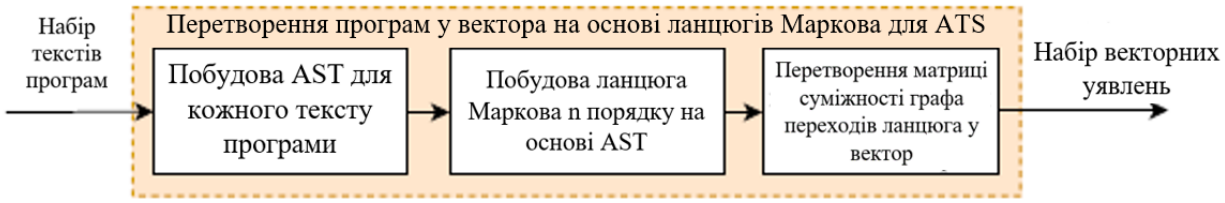


Рисунок 1.1 — Схема пропонованого методу перетворення текстів програм у векторні уявлення на основі ланцюгів Маркова для AST

Як показано на рисунку 1.1, у запропонованому методі реалізації відображення g:X→Rm пропонується спочатку побудувати AST для тексту програми або фрагмента тексту програми, який необхідно перетворити на векторне уявлення, а потім виконати побудову ланцюга Маркова порядку nn на основі AST. Після цього матриця суміжності графа переходів між станами ланцюга перетворюється на вектор.

Ланцюгом Маркова з дискретним часом [5] називають послідовність залежних дискретних випадкових величин $x_0,x_1,x_2,\ldots,x_t,x_{xt+1}$, якщо умовний розподіл x_{t+1} залежить

$$P(x_{t+1|xt,xt-1},x_{t-2},\ldots,x_0) = P(x_{t+1|xt}), \quad (1.1)$$

де x_{t+1} і x_t – величини, що йдуть один за одним, t – момент часу.

Ланцюгом Маркова з дискретним часом порядку nn називають послідовність залежних дискретних випадкових величин $x_0,x_1,x_2,\ldots,x_t,x_{xt+1}$, якщо умовний розподіл x_{t+1} залежить тільки від n $x_t,x_{t-1},x_{t-2},\ldots,x_{t+1-n}$:

$$\mathrm{P}(x_{t+1}|x_t,x_{t-1},x_{t-2},\ldots,x_0)=\mathrm{P}(x_{t+1}|x_t,x_{t-1},x_{t-2},\ldots,x_{t+1-n}), \qquad (1.2)$$

де x_{t+1} і xt – величини, що йдуть один за одним, t – момент часу, а nn – порядок ланцюга.

Значення, які можуть приймати випадкові величини $x_0,x_1,x_2,\ldots,x_t$, називають простором станів ланцюга Маркова [9], простір станів представлений у вигляді кінцевої множини $H=\{h_1,h_2,\ldots$ можливих станів ланцюга. Імовірності переходів між станами (h_i,h_j) з H можуть бути представлені у вигляді матриці $\mathrm{p}\in\mathrm{R}r\times r$. Так, для ланцюга Маркова 1-го порядку елементи матриці pp визначені як:

$$p_{ij}=\mathrm{P}\left(x_{t+1}=h_j|x_t=h_i\right),\ 0\le p_{ij}\le 1,\ \forall i:\Sigma_{j=1}^{r}p_{ij}=1,\ i=\overline{1,r}, \qquad (1.3)$$

де xt_{+1} і xt – випадкові величини, що йдуть один за одним, t – момент часу, h_i і h_j – стану ланцюга Маркова, p_{ij} -– комірка матриці pp, що характеризує ймовірність переходу h_j, r – число можливих станів ланцюга [9], матриця $\mathrm{p}\in\mathrm{R}r\times r$ є матрицею суміжності графа переходів між станами ланцюга.

У запропонованому підході до перетворення програм у векторні уявлення при побудові ланцюга Маркова для AST простір станів. Для побудови ланцюга Маркова порядку n на основі AST вводиться допоміжний алгоритм 1, що склеює пов'язані пари вершин AST, число вершин в наборах, що отримуються в результаті склеювання, дорівнює порядку ланцюга Маркова [6].

Передбачається, що вершини поданого на вхід алгоритму 1 AST представлені у вигляді кортежів, можливо, містять єдиний елемент, для

забезпечення можливості застосування алгоритму (рисунок 1.1) до результату його роботи nn раз в процесі побудови ланцюга Маркова порядку nn. Алгоритм виконує обхід безлічі переходів між вершинами AST E та виконує злиття кортежів пов'язаних вершин [6].

Алгоритм 1 має складність O ($|E|$ ^{2}), де E – число переходів між вершинами до AST, що подається на вхід. При побудові ланцюга Маркова порядку n Алгоритм 1 необхідно застосувати до результату його роботи ($n-1$) раз при ? 1 . Звідси слід, що при побудові ланцюги Маркова 1-го порядку на основі AST Алгоритм 1 не використовується, а при побудові ланцюги Маркова порядку 2 і вище складність злиття пов'язаних вершин AST оцінюється як O ($n \mid E \mid ^2$) , $n \geq 2$.

Вхід: $G = (V, E)$ D AST, де вершини представлені в вигляді упорядкованих наборів.

1. $V' = \emptyset$. D Порожнє безліч вершин зміненого AST.
2. $E' = \emptyset$. D Порожнє безліч переходів між вершинами зміненого AST.
3. **Для кожного** $(v_s, v_m) \in E$ **виконувати** :
4. **Для кожного** $(v^*, v_d) \in E$ **виконувати** :
5. **Якщо** $v_m = v^*$ **виконати** :
6. $v'_s = v_s \cup (v_{m,i})$, де $i = |v_m|$, $v_{m,i}$ - останній елемент кортежу v_m .
7. $v'_d = v_m \cup (v_{d,i})$, де $i = |v_d|$, $v_{d,i}$ - останній елемент кортежу v_d .
8. $V' \leftarrow V' \cup \{v'_s, v'_d\}$. D Додати вершини в безліч V' .
9. $E' \leftarrow E' \cup \{(v'_s, v'_d)\}$. D Додати новий перехід в безліч E' .
10. **Кінець якщо** .
11. **Кінець циклу** .
12. **Кінець циклу** .
13. **Повернути** $G' = (V', E')$ D AST, де зроблено злиття пов'язаних вершин.

Рисунок 1.2 – Злиття пов'язаних вершин AST (алгоритм 1)

Результат злиття пов'язаних вершин алгоритму 1 для AST, показаного на рисунку 1.4б, побудованого для простої програми (рисунок 1.4а),

представлений на рисунку 1.3.

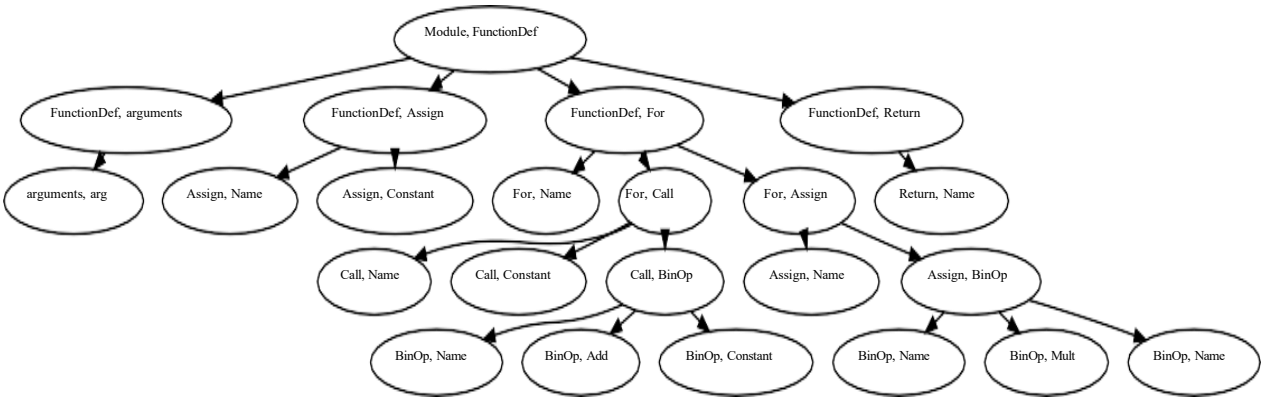


Рисунок 1.3 – Результат злиття пов'язаних вершин AST

Результат злиття пов'язаних вершин AST, показаного на рисунку 1.3 алгоритм 2 описує процес побудови ланцюги Маркова для AST $G = (V, E)$, поданого на вхід: Дерево (рисунок 1.2), в якому вершини представлені парами типів вершин відповідного . AST (рисунок 1.3в), може використовуватися при побудові ланцюги Маркова порядку 2. При цьому передбачається, що для побудови ланцюги Маркова на основі AST порядку 1 застосування алгоритма 1 не потрібно і на вхід алгоритму 2 може бути подано, наприклад, AST (рисунок 1.3в), об'єктна модель якого показано на рисунку 1.3б.

Алгоритм 2 приймає на вхід програми AST G і порядок ланцюга Маркова n. Попереднє побудова AST G здійснюється при допомоги інструментів, специфічних для мови програмування . Так, наприклад, в випадку мови програмування Python для побудови AST може використовуватися стандартна функція parse [6].

Перед побудовою ланцюга Маркова для AST виконується злиття пов'язаних вершин Алгоритму 2 в у випадку, якщо порядок ланцюга ≥ 2 (алгоритм 2). Складність цієї операції оцінюється як $O(n|E|2$), оскільки складність Алгоритму 7 оцінюється як $O(|E|2)$. Після цього в Алгоритмі 2 визначається відображення p: $V \rightarrow T$, що дозволяє отримати тип $t \in T$ конкретної вершини $v \in V$ AST. У випадку мови Python в якості відображення

p: $V \rightarrow T$ може використовуватись стандартна функція type, що має обчислювальну складність $O(1)$.

Вхід: $G = (V, E)$ D AST програми для перетворення в ланцюг Маркова.
$n \in$ N D Порядок ланцюги Маркова для AST, $n \geq 1$.
1. **Для кожного** $i = \{2, \ldots, n\}$ **виконувати** :
2. $G \leftarrow$ **Алгоритм 7** (G).
3. **Кінець циклу** .
4. **Визначити** $p: V \rightarrow T$, що відображає вершину $v \in V$ в її тип $t \in T$.
5. $M = \emptyset$. D Безліч переходів ланцюги Маркова порядку n .
6. $T = \{ g(v) : v \in V \}$. D множина типів вершин, зустрічаються в V .
7. **Для кожного** типу вершини $t \in T$ **виконувати** :
8. $V_d = \{v_d : (v_s, v_d) \in E \wedge g(v_s) = t\}$. D множина нащадків вершин типу t.
9. $T_d = \{ g(v_d) : v_d \in V_d \}$.
10. **Для кожного** типу нащадка $t_d \in T_d$ **виконувати** :
11. $\omega \leftarrow 1\{ v : v \in V \wedge g(v) = t\}|$. D Ймовірність наявності зв'язку між t та t.
$|\overline{V}_d|$ d d d d $d\,d$
12. $M \leftarrow M \cup \{(t, t_d, \omega)\}$. D Додавання нового переходу з вагою ω .
13. **Кінець циклу** .
14. **Кінець циклу** .
15. **Повернути** $G_M = (T, M)$ D Граф переходів між станами ланцюги Маркова.

Рисунок 1.4 – Побудова ланцюги Маркова порядку n для AST (алгоритм 2)

Алгоритм 2 формує множина T, що складається з типів вершин, що зустрічаються в AST, на основі обходу вершин AST, складність цієї операції дорівнює $O(|V|)$. Після цього для кожного типу $t \in T$ виконується пошук дочірніх вершин $V\ d$ для вершин типу t, для цього виконується обхід безлічі переходів між вершинами E , складність якого дорівнює $O(|E|)$ на основі $V\ d$ формується безліч типів нащадків $T\ d$, складність цією операції дорівнює $O(|Vd|)$. Для кожного з типів нащадків $t\ d$ вершин типу t обчислюється ймовірність наявності переходу з t до $t\ d$, складність цієї операції дорівнює $O(|Td\ ||Vd\ |)$.

Таким чином, складність алгоритма 2 в випадку, якщо порядок ланцюги Маркова $n \geq 2$, складає $O(n|E|2+|V|+|T||E|+|T|bc)$,

де $|E|$ – число переходів між вершинами в AST,

$|V|$ – число вершин в AST,

$|T|$ – число типів вершин, зустрічаються в AST,

b – усереднене число нащадків вершин типу t у множині $V\ d$,

c – середнє число типів нащадків вершин типу t в множині Td.

Складність побудови ланцюгів Маркова 1-го порядку по алгоритму 2 дорівнює O ($|V|$ + $|T||E|$ + $|T|$ bc), оскільки алгоритм 1 у разі не використовується. Результатом роботи алгоритма 2 є зважений граф переходів GM між станами ланцюга Маркова, ваги ребер якого відповідають ймовірностям зв'язків між вершинами AST. На рисунку 1.3а та рисунок 1.3б показані результати застосування алгоритму 2 до AST програм мовою Python, представлених на рисунок 1.4а та рисунок 1.4б відповідно. У ланцюгу Маркова на рисунку 1.5б присутні оператори For і Assign , відсутні в ланцюги Маркова на рисунку 1.5а, також на рисунок 1.3б зв'язку між вершиною FunctionDef та вершинами Return, argu - ments мають меншу вагу через наявність додаткового зв'язку між FunctionDef та оператором Assign.

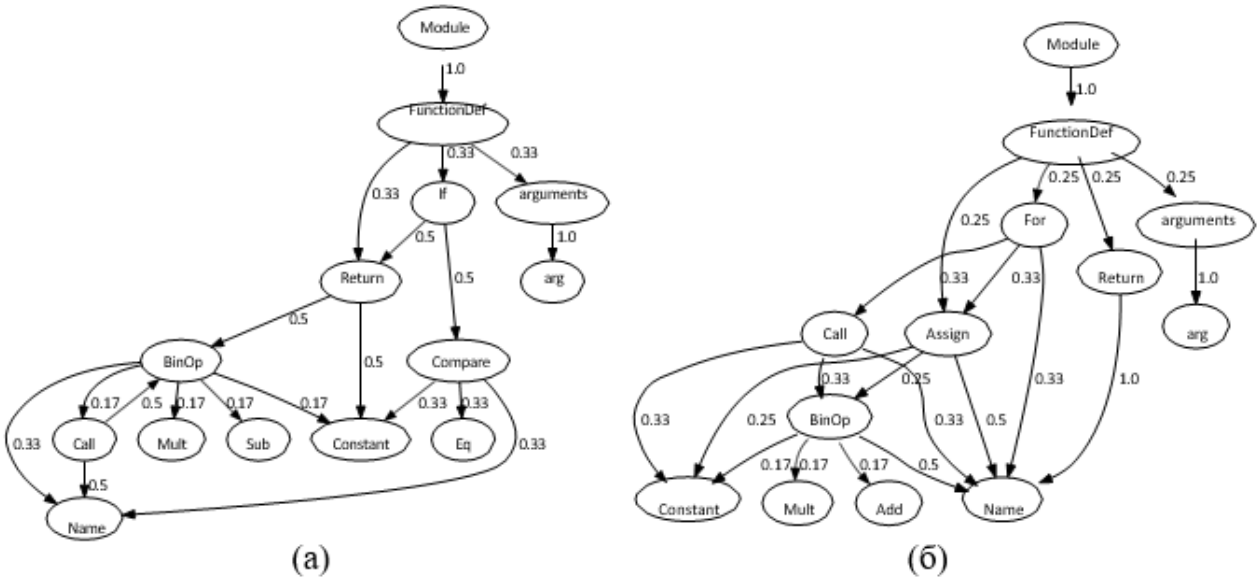


Рисунок 1.5 – Побудова Ланцюга Маркова

Ланцюг Маркова 2-го порядку, побудована для програми, текст якої показаний на рисунок 1.5б, наведено на рисунку 1.5. AST для цієї програми показано на рисунку 1в, а результат застосування алгоритму 2 до AST показаний на рисунок 2.2.

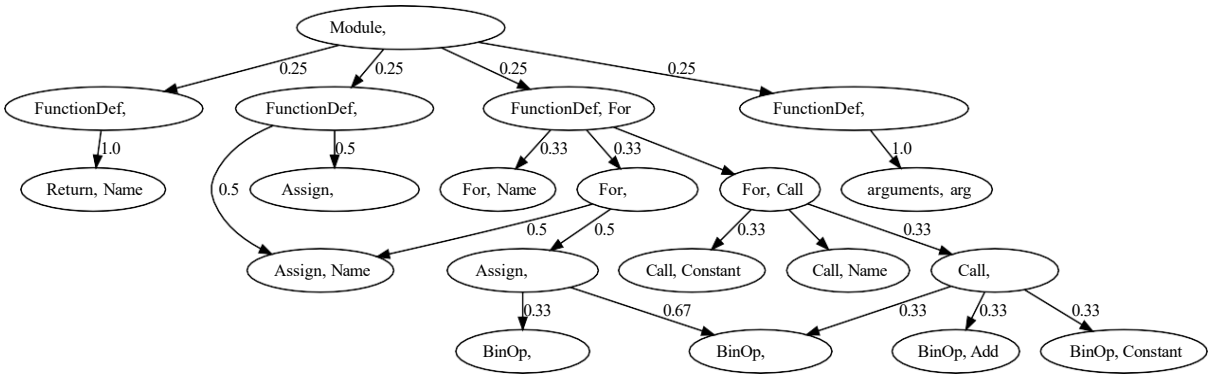


Рисунок 1.6 – Ланцюг Маркова 2-го порядку

Як видно з рисунка 1.5б та рисунок між станами ланцюги Маркова для AST, число зв'язків при цьому збільшується незначно. Так, у ланцюзі Маркова 1-го порядку, побудованої для програми, показаної на рис. 13 вершин і 20 зв'язків (рисунок 1.5б), а в ланцюги Маркова 2-го порядку – 20 вершин і 21 зв'язок. порядку ланцюги n наводить до зростання часу роботи алгоритма 2 з тимчасовий складністю $O(|E|2 + |V| + |T||E| + |T|\ bc\ )$ при n = 2, складністю $O(|V| + |T||E| + |T|bc\ )$ при n=1.

Побудова векторного представлення *i-ї* програми на основі ланцюга Маркова порядку n для AST здійснюється перетворенням матриці суміжності $p_i \in R^{h \times h}$ зваженого графа переходів між h станів ланцюга Маркова у вектор $\vec{v}$? Так, для графа G^1 , показаного на рисунку 1.3а і містить 14 вершин, матриця суміжності котрого належить простору $R^{14 \times 14}$, може бути побудований вектор R^{196} . Для графа G^2, показаного на рисунку 1.3б і містить 13 вершин, матриця суміжності якого належить $R^{13 \times 13}$, може бути побудований вектор $\vec{v}_2 \in R^{169}$, число компонентів вектора $\vec{v}_1$ не дорівнює числу компонент M^2.

У випадку, якщо необхідно перетворити набір AST програм в набір векторних уявлень, що належать простору R m, пропонується використовувати Алгоритм 9, що здійснює побудову безлічі H, містить типи всіх вершин, що зустрічаються в AST з набору G , подається на вхід алгоритму. Після побудови множини H обчислюється його потужність h

(рядок 9).

Вхід: $G = \{ G^1, G^2, \dots, G^r \}$ D Набір AST програм для перетворення в векторів.
$n \in N$ D Порядок ланцюги Маркова для AST, $n \geq 1$.

1. $H = \emptyset$. D Безліч типів вершин AST, зустрічаються в ланцюгах Маркова.
2. $R = \emptyset$. D Упорядкований набір множин переходів між вершинами ланцюгів Маркова.
3. **Для кожного** AST $G^i \in G$ **виконувати** :
4. $G^i \leftarrow$ **Алгоритм 2** (G^i, n). D Побудувати ланцюг Маркова $G^i = (T^i, M^i)$ порядку n .
5. $H \leftarrow H \cup T^i$. D Об'єднати безліч вершин ланцюги Маркова T^i з безліччю H .
6. $R \leftarrow R \cup (M^i)$. D Додати безліч зв'язків між вершинами M^i в набір R .
7. **Кінець циклу** .
8. $B = \emptyset$. D Упорядкований набір векторних уявлень ланцюгів Маркова.
9. $h = | H |$. D Число типів вершин AST, зустрічаються в ланцюгах Маркова.
10. **Для кожного** безлічі зв'язків між вершинами ланцюги Маркова $M^i \in R$ **виконувати** :
11. **Перетворити** матрицю суміжності $\mathbf{p}_i \in R^{h \times h}$ графа $G^i = (H, M^i)$ до $\vec{v} \in R^m$, $m = h^2$.
12. $B \leftarrow B \cup (\vec{v}^*_i)$. D Додати векторне подання програми $\vec{v}^*_i$ в набір B .
13. **Кінець циклу** .
14. **Повернути** (B, H) D Набір векторів програм, безліч типів вершин з AST.

Рисунок 1.7 –Перетворення набору AST в набір векторів на основі ланцюгів Маркова

Потім кожна і -я ланцюг Маркова $G_i = (T_i , M_i)$, де T_i – множини вершин і ланцюги, M_i – множина зв'язків між вершинами і -ї ланцюги, перетворюється в граф $G_i = (H, M)$. У результаті множини вершин кожною ланцюги замінюється на множину H перед побудовою матриці суміжності (алгоритм 3). Після виконання цієї операції кожна матриця суміжності p_i і -го графа $G\ i$ належить матричному простору R $h \times h$, де $h = |H|$. Кожна матриця $p_i \in$ R $h \times h$ конкатенацією рядків перетворюється на вектор $\vec{v}i \in Rm$, де $m = h_2$, таким чином AST всіх програм перетворюються до векторним уявленням, що належать векторному простору однієї й тієї ж розмірності m.

1.3 Визначення завдання по тексту програми

Для експериментальної оцінки якості класифікаторів KNN, SVM, RF, ШНМ [6], на вхід яким подаються векторні уявлення текстів програм на основі ланцюгів

Маркова для AST, використовувався відкритий набір даних [5], що містить текст програми 13881. Програми в наборі [5] вирішують унікальні завдання 11 різних типів, що автоматично згенеровані системою Цифровий асистент викладача (ЦАВ) [15].

У процесі оцінки якості наважувалася завдання багатокласовий класифікації, рішенням який є побудова класифікуючого відображення a: $X \rightarrow Y$, здатного визначати тип завдання $y\ i \in Y$ при Y = {1,2, … ,11} за текстом програми $x\ i \in X$. Набір даних для аналізу був заданий кінцевою множиною $\{(x_i\ ,\ y_i) : x\ i \in X,\ y\ i \in Y\}$, що містить 13881 пар, що включають текст програми $x\ i \in X$ і тип розв'язуваної програмою задачі $y\ i \in Y$. Класифікатори на основі KNN, SVM, RF, ІНС приймають вектори на вхід, тому програми з множини X були попередньо перетворені на векторні уявлення, що належать R m , на основі різних способів реалізації відображення g: $X \rightarrow$ R m , включаючи підходи на основі ланцюгів Маркова 1-го і 2-го порядків для AST (алгоритм 1 і алгоритм 2) [7], підхід на основі word2vec і code2vec [4], гістограми кодів операцій (алгоритм 1) [6], гістограми типів вершин AST (алгоритм 2), причому розмірність цільового векторного простору R залежала від способу побудови векторних уявлень.

Програми з відкритого набору [15] написані на мовою Python. Генератори унікальних завдань 11 типів, використовувані в ЦАВ, засновані на методі рішення завдання задоволення обмеженням «згенерувати і перевірити». Вирішення 13881 унікальних завдань з відкритого набору програм [5] були надіслані в групу тестувальників, автоматично перевірені і прийнято. Число програм, згрупованих на вид розв'язуваних завдань, показано на рисунку 1.6.

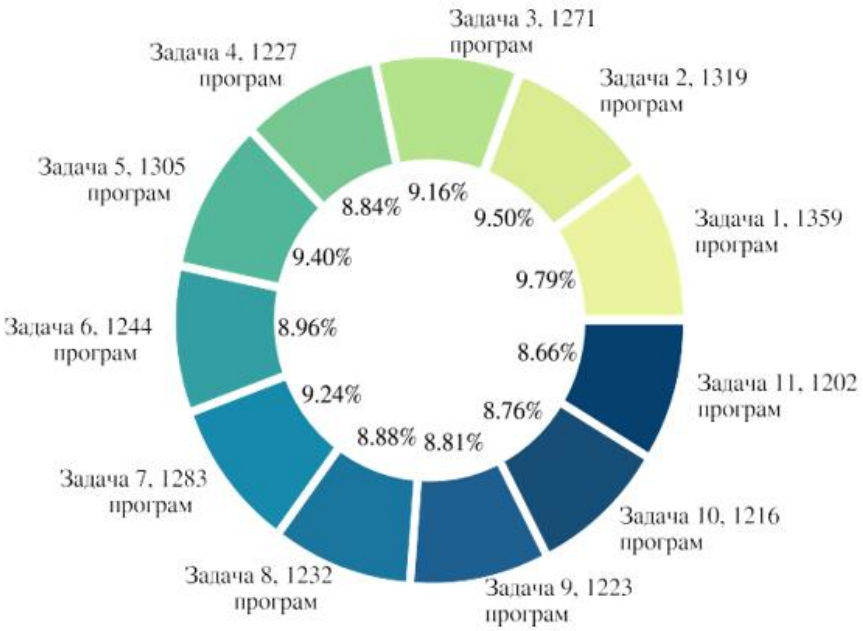


Рисунок 1.8 – Число програм, згрупованих по типам завдань

Кожне унікальне завдання заданого типу, згенерована ЦАВ, могла бути вирішена з використанням будь-якого алгоритму, котрий вибере що навчається. Приклади програм з набору даних [5], вирішальних завдання різних типів з використанням різняться алгоритмів, наведено в таблиці 1.1. У таблиці виділено основну відмінність задачі визначення розв'язуваної програмою завдання від завдання визначення реалізованого алгоритму. Відповідно до таблиці 1.1, одне й те завдання може бути вирішена під час використання різних алгоритмів.

Як показано у таблиці 1.1, перший підхід до реалізації рекурентної формули використовує цикл, а другий підхід – рекурсивну функцію. Перший підхід до реалізації розбору двійкового формату використовує клас із прихованим станом, а другий підхід – функцію, яка повертає розібране значення та наступне зміщення курсору [5].

Таблиця 1.1 – Приклади програм, вирішальних завдання типу 4 і типу 11 різними способами

Завдання	Алгоритм 1	Алгоритм 2

4	def main(n):
	res1 = - 0.31
	res2 = - 0.44
	for i in range(1, n):
	temp = res1 - res2 ** 3 / 38 res1 = res2
	res2 = temp
	return temp
		def main(n):
		if n == 0:
		return 0.21
		if n >= 1:
		ans = main(n - 1) + 1
		ans += math.atan (main(n - 1)) ** 3
		return ans

11	…
	def read_ a (reader: BinaryReader):
	a1 = reader.read_uint16()
	a2 = read_ array (source = reader.source ,
	size=reader.read_uint32(),
	address=reader.read_uint32(), read= lambda reader: read_b (reader),
	structure_size = 50)
	a3 = read_d (reader)
	a4 = [reader.read _int32()]
	return dict (A1=a1, A2=a2, A3=a3, A4=a4)
	…
		…
		def parse_ c (buf , offs):
		c1, offs = parse(buf , offs, 'float') c2, offs = parse(buf , offs, 'float') c3, offs = parse (buf , offs, 'uint16')
		c4size, offs = parse (buf , offs, 'uint32') c4offs, offs = parse (buf , offs, 'uint16') c4 = []
		for _ in range(c4size):
		val , c4offs = parse (buf , c4offs, 'uint8') c4.append(val)
		return dict (C1=c1, C2=c2, C3=c3, C4=c4)
		…

Для оцінки якості класифікаторів, на вхід яких подаються векторні уявлення текстів програм, отримані різними способами, використовувався комп'ютер під керуванням операційної системи Windows 10 з інтерпретатором мови Python версії 3.10. Результати проведених експериментів не залежать від характеристик обладнання, так як в даному розділі оцінюється якість класифікації, а чи не продуктивність алгоритмів.

У підході до вкладенню програм в векторне простір на основі word2vec

використовувалася реалізація CBOW [6], доступна в бібліотеці Gensim, причому програми на Python були попередньо перетворені в упорядковані набори токенів з допомогою модуля tokenize [7]. Підхід до вкладенню програм в векторне простір на основі code2vec включав використання PathMiner 0.9.0 [8], публічно доступного в вигляді образу Docker з ім'ям voudy/astminer, реалізація IHC code2vec була запозичена з [4] та адаптована для підтримки Windows. Підходи до перетворення програм в векторні уявлення на основі ланцюгів Маркова і на основі гістограм були реалізовані на Python внаслідок їх простоти , у реалізаціях використовувалися модулі ast [6] та dis [5].

Реалізації алгоритмів класифікації, таких як KNN, SVM, RF та ШНМ, були взяті з Python – бібліотеки sklearn [10]. У ході проведених експериментів застосовувалася перехресна перевірка по k блоків во час запуску тестів для кожною пари з класифікатора і підходу до вкладення програм в векторне простір для перевірки якості класифікатора на різних частинах вихідного набору даних. Рисунок 1.9 визначає принцип роботи перехресної перевірки по k блокам.

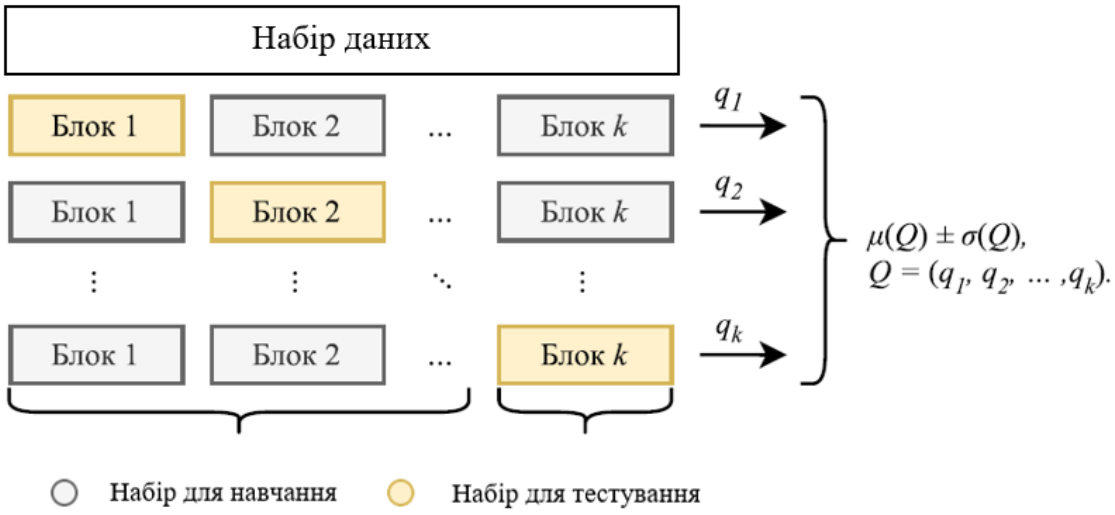


Рисунок 1.9 -– Перехресна перевірка по k блоків

Як показано на рисунку 1.9, при перехресний перевірці по k блоків якість класифікатора вимірюється шляхом поділу вихідного набору даних на k блоків k разів. Для кожного поділу принавчанні класифікатора використовується k –1

блоків, а 1 блок використовується для оцінки якості з користуванням спеціалізованою заходи якості, такий як загальна точність (1.18), точність (1.15), повнота (1.16) або F_1 -міра (1.17). Блоки, використовувані во час навчання, виділено сірим кольором на рисунку 1.6, а блоки, які використовуються для оцінки якості, виділені помаранчевим кольором. Середнє значення μ і стандартне відхилення σ для набору Q на рисунку 1.6 визначаються як

$$\mu(Q) = \frac{1}{k}\sum_{i=1}^{k} q_i, \tag{1.4}$$

$$\sigma(Q) = \sqrt{\frac{1}{k}\sum_{i=1}^{k}\left(q_i - \mu(Q)\right)^2}. \tag{1.5}$$

Рисунок 1.10 описує процес класифікації програм для збирання експериментальних оцінок пар, що складаються з класифікатора та підходу до вкладення програм у векторний простір.

Рисунок 1.10 – Класифікація текстів програм для збору експериментальних оцінок

Класифікація текстів програм для збору експериментальних оцінок показаний на рисунку 1.10, включає перетворення програм в векторні уявлення, розбиття вихідного набору програм (рисунок 1.6) на k блоків, при цьому кожен з k раз виконується навчання і тестування класифікатора (рисунок 1.8) для отримання множини оцінок якості Q, $|Q| = k$.

2 МЕТОДИ І АЛГОРИТМИ ВИНЕСЕННЯ РЕКОМЕНДАЦІЙ ПО ПРИСКОРЕННЯ ПРОГРАМ

2.1 Винесення рекомендацій по підвищення швидкості виконання програм

У сучасних гетерогенних обчислювальних платформах CPU спільного призначення доповнюється спеціалізованими прискорювачами, такими як GPU [9], ASIC, FPGA [10]. Існує потреба в прискорення ПС на основі можливостей, наданих заданою гетерогенної обчислювальної платформою, для досягнення найкращою швидкості роботи ПС при виконанні обчислень [9]. У цьому актуальна розробка методів та алгоритмів ідентифікації фрагментів текстів програм ПС в процесі статичного аналізу для винесення рекомендацій щодо них прискорення на заданою гетерогенної обчислювальної платформі, де функціонує ПС [6]. Дотримання таких рекомендацій дозволить підвищити швидкість виконання фрагментів текстів програм і, як наслідок, призведе до підвищення швидкості роботи ПС в цілому. У цьому розділі розглядається завдання ідентифікації фрагментів текстів програм для винесення рекомендацій щодо переробки коду з метою підвищення швидкості виконання програм. Для вирішення цієї завдання пропонується методика, заснована на попередньому формуванні бази програм- прикладів і варіантів підвищення їх швидкості виконання при допомоги прискорювачів, доступних на заданій гетерогенній обчислювальній платформі, з подальшим застосуванням алгоритму пошуку фрагментів у AST за програмою-прикладом.

Пошук фрагментів текстів програм за програмою-прикладом може виконуватися для виявлення таких фрагментів програм, які володіють тими ж властивостями, що і шукана програма-приклад. Завдання ідентифікації фрагментів текстів програм для винесення рекомендацій по підвищення їх швидкості виконання на заданою гетерогенної обчислювальної платформі

може бути зведена до задачі формування бази програм-прикладів і варіантів підвищення їх швидкості виконання при допомоги прискорювачів, доступних на обчислювальної платформи, а також завдання пошуку фрагментів текстів програм за програмою-прикладом.

Запропонована методика ідентифікації фрагментів текстів програм для винесення рекомендацій по оцінки перспектив підвищення їх швидкості виконання включає наступні кроки [8].

Крок 1. Формування бази програмних реалізацій алгоритмів.

Крок 2. Трансляція реалізацій алгоритмів в уявлення, придатні для запуску на кожному з прискорювачів, доступних на заданій гетерогенній обчислювальній платформі, де розгортаються та експлуатуються ПС.

Крок 3. Оцінка часу роботи програмних реалізацій алгоритмів на кожному з прискорювачів, доступних на заданій гетерогенній обчислювальній платформі.

Крок 4. Доповнення бази програмних реалізацій алгоритмів коефіцієнтами, обчисленими на основі оцінок швидкості виконання реалізацій алгоритмів на доступних на заданою гетерогенної обчислювальної платформи прискорювачах.

Крок 5. Виконання пошуку фрагментів AST за програмою-прикладом у текстах програм, що аналізуються ПС для кожної з програмних реалізацій алгоритмів у базі програмних реалізацій алгоритмів.

Крок 6. Створення звіту о перспективах прискорення ПС.

У звіт, формується на шостому кроку запропонованої методики [8], включаються знайдені фрагменти ПС, і навіть прогнозовані коефіцієнти їх прискорення, отримані кроках 2-4.

Завдання пошуку фрагментів текстів програм за програмою-прикладом, що вирішується на п'ятому кроці методики, представлена трійкою (G_t, G_q, A), де $G_t \in G$ – AST для статичного аналізу, $G_q \in G$ – AST, що представляє собою пошуковий запит, $A = \{a_1, a_2, \dots, a_n\}$ кінцеві множини з n правильних відповідей, тобто таких фрагментів G_t, які насправді відповідають запиту

G_q, причому $\forall a_i \in A$: $a_i \in G$, а G — безліч AST G_q, A метою є побудова відображення g: $G \times G \to \{G\}$, яке ставить у відповідність аналізованому AST G_t та запитом G_q безліч з k фрагментів $P = \{p_1, p_2, \ldots, p_k\}$, знайдених по запиту G_q, причому $P \in$

$\{G\}$, де $\{G\}$ - множина всіх множин AST G, причому $\forall p_i \in P$: $p_i \in G$, а для оцінки якості k результатів пошуку безліч P порівнюється з безліччю правильних відповідей A.

При рішенні завдання ідентифікації фрагментів текстів програм для винесення рекомендацій щодо підвищення їх швидкості виконання пошук фрагментів за прикладом виконується для виявлення коду, прискорена версія якого прискорюється по порівнянні з оригінальною версією во стільки ж раз, во скільки прискорюється програма-приклад. Процедура пошуку фрагментів текстів програм з програмі-прикладу повторюється для кожною програми, включеною в базу програм-прикладів та варіантів підвищення їх швидкості виконання при допомоги прискорювачів, доступних на спеціалізованою гетерогенної обчислювальної платформі. Звіт о перспективах прискорення ПС пропонується формувати на основі результатів пошуку, отриманих на п'ятому кроку методики, а також на основі коефіцієнтів, обчислених на четвертому кроку методики по результатам оцінок швидкості виконання реалізацій алгоритмів із підготовленої бази їх програмних реалізацій.

2.2 Методи і алгоритми пошуку фрагментів текстів програм по програмі-прикладу

У розглянутій задачі (G_t, G_q, A) пошуку фрагментів у AST G_t за програмою-прикладом G_q множини правильних відповідей A, як правило, містить невелику кількість фрагментів, що відповідають програмі-прикладу G_q. Отже, завдання зводиться до розробці алгоритму, стійкого до незбалансованих даних. У процесі рішення завдання (G_t, G_q, A) пропонується використовувати метод найближчого сусіда, відстань Йенсена-Шеннона, а

також перетворити програму-приклад G_q і фрагменти, вилучені з AST G_t , в векторні представлення на основі ланцюгів Маркова, побудова яких виробляється та AST, і для графа «Визначення-використання». Крім того, пропонується порівнювати цикломатичну складність [6] фрагментів, що витягуються з AST G_t, з цикломатичною складністю програми-прикладу G_q.

v D Аналізована вершина AST.

s D Словник, що містить змінні і пов'язані з ними вершини AST.

$V = \emptyset$, $E = \emptyset$.

Якщо v є функцією, **виконати:**

$s \leftarrow \emptyset$. D Помістити в s порожній словник.

Кінець якщо.

Для кожною змінної η , яку *використовує* вершина v , **виконати** :

$E \leftarrow E \cup \{(s[\eta], v)\}$.

$V \leftarrow V \cup \{s[\eta]\} \cup \{v\}$.

Кінець циклу .

Для кожною змінної η , яку *визначає* вершина v , **виконати:**

Якщо v є привласненням, **виконати:**

$v \leftarrow v_{\text{rhs}}$, де v_{rhs} - права частина присвоєння, дочірня вершина для v .

Кінець якщо .

$s[\eta] \leftarrow v$. D Помістити в словник s вершину v по ключу η .

Кінець циклу .

Для кожною вершини v_c , дочірньої вершини для v , **виконати** :

$E \leftarrow E \cup$ **Алгоритм 11** (v_c, s) .

Кінець циклу .

Повернути (V, E) . D DU- граф.

Рисунок 2.1 – Побудова графа «Визначення-використання» на основі AST (алгоритм 4)

DU-граф – це спрямований ациклічний граф, що з'єднує місця визначення змінних з місцями їх використання. DU-графи широко застосовуються для аналізу потоків даних програм в компіляторах і статичних аналізаторах [10]. У даної роботі місцем визначення або використання змінної в DU-графі пропонується вважати вершину v , яка містить посилання на

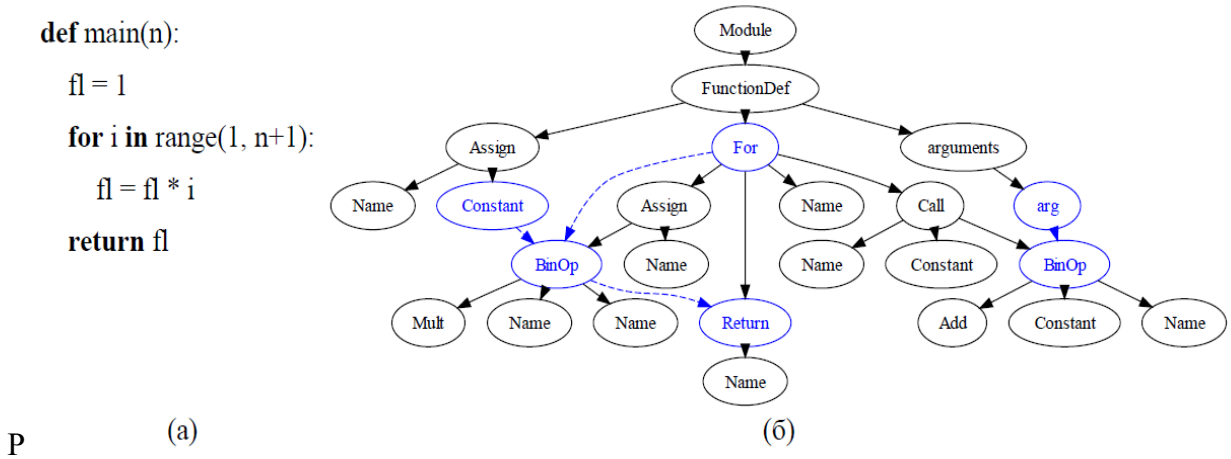
визначувану або використовувану змінну, причому в випадку оператора присвоєння місцем визначення змінної вважається права частина присвоєння v_{rhs}. Увімкнення інформації о залежно за даними, представленими переходами між вершинами DU-графа, у векторні представлення програм на основі ланцюгів Маркова, дозволить виявляти відмінності між синтаксично схожими програмами, але з потоками даних, що істотно розрізняються.

Алгоритм 4 описує процес побудови DU-графії. на вхід Алгоритму 3 подається корінь AST, при цьому передбачається, що кожна вершина v AST містить посилання на дочірні вершини v_c . Наприклад, вершини AST мови Python, побудова якого виробляється засобами [9], а також вершини AST мови C#, побудова якого виробляється засобами Microsoft™ Roslyn® [10] містять іменовані посилання на дочірні вершини.

Алгоритм 4 виконує рекурсивний обхід AST програми, починаючи з поданою на вхід вершини v. Побудова AST пропонується виконувати засобами, залежними від використовуваного мови програмування. Так, наприклад, в випадку Python для побудови AST може використовуватися [6], а у разі C# – API Microsoft Roslyn [2]. Квадратні дужки на рядках 6 і 13 в алгоритмі 4 використовуються для позначення запису елемента в словник ключу і отримання елемента з словника по ключу відповідно. У процесі обходу AST Алгоритмом 4 на основі інформації про визначення та використання змінних формується багатожінство вершин DU-графа V та множина переходів між вершинами DU-графа E. Складність алгоритма 4 в середньому оцінюється як $O\ (|V|\varepsilon)$, де $|V|$ – число вершин у аналізованому AST, ε – середня кількість змінних, які використовує або визначає кожна вершина AST.

Для докладнішого розгляду алгоритму 4 на рисунку 2.2а представлена проста програма, що обчислює факторіал, а на рисунку 2.2б чорним кольором показано AST цією програми, побудоване коштами стандартною бібліотеки мови Python [6]. У результаті обходу AST по алгоритму 4 будується DU-граф, що містить переходи між вершинами AST, що визначають або використовують змінні з одним і тим самим ім'ям (рисунок 2.2б). Сині

пунктирні стрілки на рисунку 2.2б позначені переходи, що належать DU-графу, побудованому за алгоритмом 4, а чорними суцільними стрілками позначені переходи, що належать до AST. Синім кольором на рисунок 2.2б позначені вершини, що належать як AST, так і DU-графу, а чорним кольором позначені вершини, що належать лише AST.



Р

Рисунок 2.2 — Програма, її AST та DU-граф : а) проста програма мовою Python, яка обчислює факторіал; б) AST програми, в якому синім кольором позначені вершини та переходи, що належать DU-графу, побудованому за алгоритмом 4

Як показано на рисунку 2.2б і вказано на рядках 10 і 11 в Алгоритм 4, в випадку вершини Assign місцем визначення змінної вважається права частина присвоєння v_{rhs}, в інших випадках місцем визначення або використання змінної вважаються вершини AST, є батьками вершин Name, містять імена визначених або використовуваних змінних.

Ланцюг Маркова, побудована на основі AST, показаного на рисунку 2.3б, наведено на рисунку 2.3а, а ланцюг Маркова, побудована на основі AST і DU-графа, показано на рисунку 2.3б. Синіми пунктирними стрілками на рисунок 2.2б позначені переходи, належать ланцюги Маркова, побудованою на основі DU-графа, переходи між вершинами якого також позначені синім пунктиром на рисунок 2.2б. Чорними суцільними стрілками на рисунку 2.3б

позначені переходи, належать ланцюги Маркова, побудована на основі AST. Синій колір на рисунку 2.3б позначені вершини, що належать як ланцюги Маркова для AST, так і ланцюги Маркова для DU-графа, а чорним кольором позначені вершини, що належать тільки Маркова ланцюга для AST.

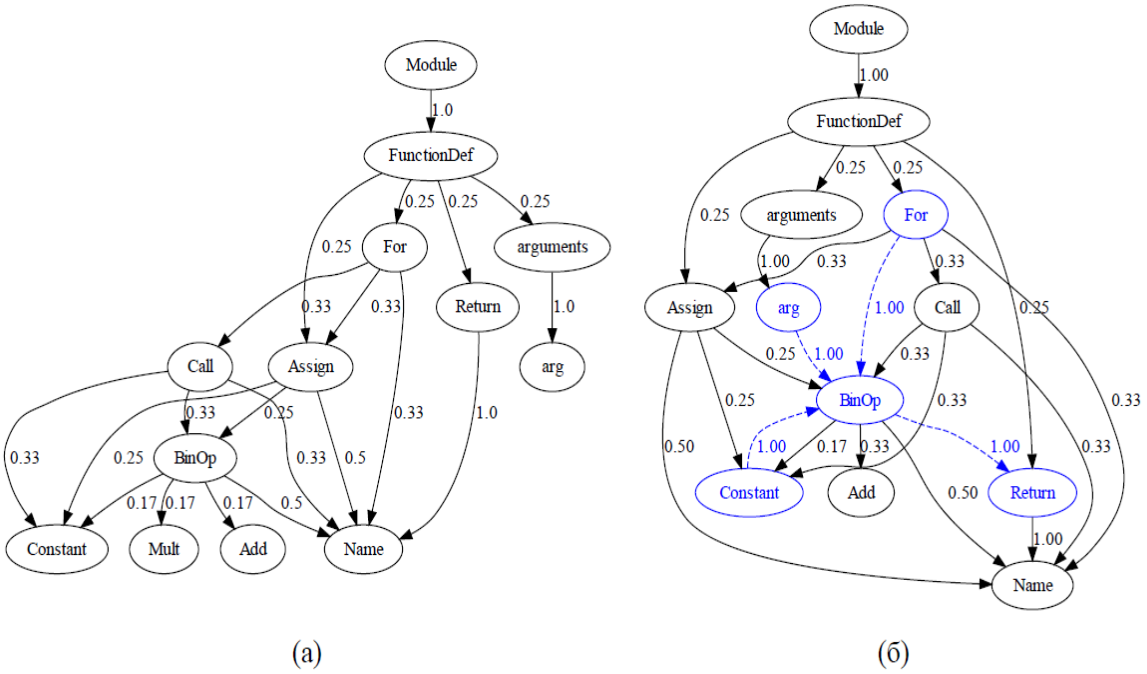


Рисунок 2.3 – Ланцюг Маркова на основі AST і DU-графа для програми

Крім того, пропонується також оцінити цикломатичну складність [6] вилученого фрагмента програми і цикломатичну складність програми-прикладу, порівняти отримані оцінки цикломатичної складності та включити результат порівняння до підсумкової міри відстані між програмами. Програми з різною цикломатичною складністю містять різне число циклів та операторів розгалуження [6], а число циклів, у тому числі вкладених, впливає на швидкість виконання програм.

2.3 Оцінка якості векторних уявлень текстів програм на основі ланцюгів Маркова

Оцінка якості векторних уявлень текстів програм на основі ланцюгів Маркова 1-го порядку, побудова яких здійснюється і для AST, і для DU-графів

(рисунок 2.2б). Для експериментальної оцінки якості класифікаторів KNN, SVM, RF, ШНМ [6], на вхід яким подавалися як векторні представлення текстів програм на основі ланцюгів Маркова для AST (рисунок 2.2а), так і векторні представлення текстів програм на основі ланцюгів Маркова для AST і для DU-графів (рисунок 2.2б), використовувався відкритий набір даних, що містить 13881 текст.

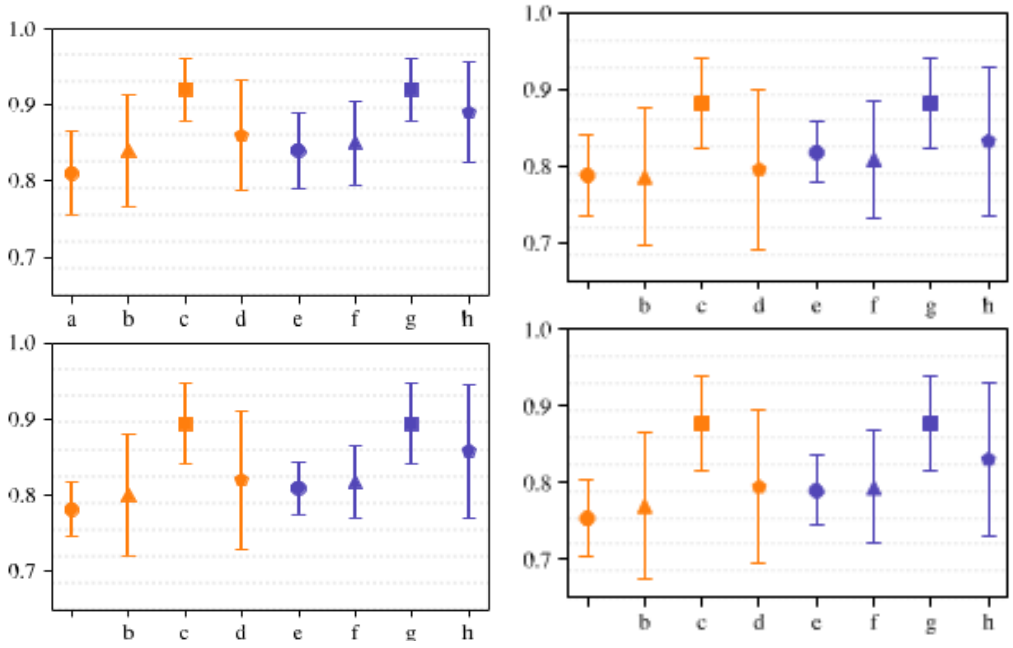


Рисунок 2.4 – Оцінки якості класифікації для пар з підходу до перетворенню програм у векторні уявлення та класифікатора для набору, що містить 100 програм

Реалізації алгоритмів класифікації, таких як KNN, SVM, RF та ШНМ, були взяті з Python-бібліотеки sklearn [10] Параметри KNN, SVM, RF, ШНМ наведені в таблиці 2.2 для кожною пари з класифікатора і підходу до вкладенню програм в векторне простір для перевірки якості класифікатора на різних частинах вихідного набору даних. Для оцінки якості класифікаторів використовувалися такі заходи якості, як загальна точність, точність, повнота і F_1 міра. Для визначення, які пари з класифікатора і підходу до вкладенню програм в векторне простір, дозволяють досягти кращою точності класифікації

для набору даних малого розміру, містить 100 програм, були отримано оцінки, показані на рисунок

Крапками різних форм на рисунку 3.3 виділено середня значення якості класифікаторів KNN, SVM, RF, MLP при їх спільному використання з різними векторними уявленнями текстів програм. Вертикальні відрізки на рисунку 2.3 позначають інтервали, обмежені зверху значенням $x + \sigma$ та знизу значенням $x - \sigma$, де $x -$ середня значення якості, а σ - СКО. Дані, показані на рисунку 2.3 також представлені в числовому вигляді в таблиці 2.1.

Таблиця 2.1 – Оцінки якості класифікації для пар з підходу до перетворення текстів програм у вектори та класифікатора, для набору, що містить 100 текстів програм [5]

Вкладення програм	Алг.	Загальна точ. (1.15)	Точність (1.16)	Повнота (1.17)	F 1 -міра (1.18)
Ланцюг Маркова 1-го порядку для AST (Див. розділ 2.1)	KNN	81.0± 5.5	78.8± 5.3	78.2± 3.5	75.4± 5.1
	SVM	84.0± 7.3	78.6± 8.9	80.0± 8.0	77.0± 9.6
	RF	92.0± 4.0	88.2± 5.9	89.4± 5.3	87.8± 6.1
	IHC	86.0± 7.3	79.6± 10.5	82.0± 9.1	79.5± 10.0
Ланцюг Маркова 1-го порядку для AST та DU-графа (Див. розділ 3.2.1)	KNN	84.0± 4.9	81.8± 4.0	80.9± 3.5	79.0± 4.6
	SVM	85.0± 5.5	80.9± 7.6	81.8± 4.8	79.4± 7.4
	RF	92.0± 4.0	88.2± 5.9	89.4± 5.3	87.8± 6.1
	IHC	89.0± 6.6	83.3± 9.7	85.8± 8.7	83.1 ± 10.0

Результати порівняння чутливості (2.7) до класифікатору векторних уявлень текстів програм на основі ланцюгів Маркова для AST або для AST та DU-графів, на наборі даних, що містить 100 текстів програм з набору, показано на рисунок 2.4.

Як показано на рисунку 2.3 і в таблиці 2.2, векторні уявлення текстів програм на основі ланцюгів Маркова, побудова яких виробляється і для AST, і для DU-графів (рисунок 2.2б), дозволяють досягти кращою точності класифікації по порівнянні з векторними уявленнями текстів програм на основі

ланцюгів Маркова тільки для AST (рисунок 3.2а).

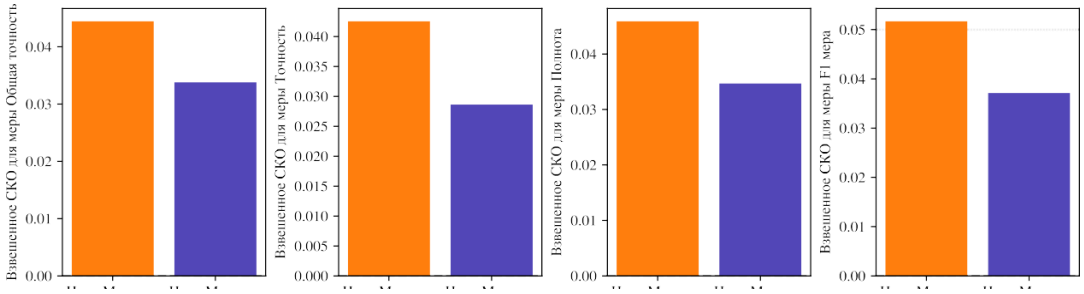


Рисунок 2.5 – Результати порівняння чутливості до використовуваного класифікатора методів перетворення програм в векторні уявлення на наборі даних із 100 текстів програм з вихідного набору

Якість класифікаторів на основі KNN та MLP покращується значно, в середньому на 3-4%, якість класифікатора на основі SVM покращується незначно, в середньому на 1-2%. Крім того, як видно із рисунок 2.4, векторні уявлення програм на основі ланцюгів Маркова, побудова яких виробляється і для AST, і для DU-графів, мають знижену чутливість до класифікатора.

Додаткові експерименти показали, що зі збільшенням числа програм у навчальному наборі різниця між якістю класифікаторів, на вхід яким подаються вектори на основі ланцюгів Маркова для AST (рисунок 3.2а), і якістю класифікаторів, на вхід яких подаються вектори на основі ланцюгів Маркова для AST і для DU-графа (рисунок 2.2б), зникає, що робить вектори програм на основі ланцюгів Маркова для AST та для DU-графа придатними для практичних застосувань в завданнях класифікації, що характеризуються малим набором зібраних даних для навчання, незалежно від обраного класифікатора.

2.4 Алгоритм пошуку фрагментів текстів програм по програмі-прикладу

Для вирішення задачі (G_t , G_q , A) також пропонується алгоритм 5, що виконує вилучення фрагментів з AST G_t і оцінку їх подібності з програмою-прикладом G_q.

Вхід : G_q D AST програми-прикладу.
$G_t = (V_t, E_t)$ D AST, в якому виконується пошук фрагментів по програмі-прикладу.
$\omega_1 \in [0,1]$ D Вага відстані між векторами на основі ланцюгів Маркова для AST.
$\omega_2 \in [0,1]$ D Вага відстані між векторами на основі ланцюгів Маркова для DU- графа.
$\omega_3 \in [0,1]$ D Вага заходи подібності цикломатичної складності програм.
ϑ D Число вершин в комбінаціях вершин, вилучених з AST.
κ D Число найкращих результатів пошуку.

1. $R = \emptyset$
.
2. **Для кожною** вершини $v \in V_t$ **виконувати** :
3. $\xi = v_{body}$. D Помістити в ξ кортеж послідовно виконуваних інструкцій v.
4. **Якщо** $\xi \neq \emptyset$ **виконати**:
5. **Для кожною** комбінації вершин AST $(v_1, v_2, \dots, v_P) \in (\xi)$ **виконувати** :
6. **Створити** тимчасове AST G з кортежем інструкцій $(v_1, v_2, \dots, v_P)$.
7. $\rho \leftarrow$ **Алгоритм 13** $(G, G_q, \omega_1, \omega_2)$.
8. $R \leftarrow R \cup \{(\rho, G)\}$.
9. **Кінець циклу** .
10. **Кінець якщо**.
11. **Кінець циклу**.
12. **Повернути** множина P , містить κ пар з мінімальним відстанню ρ з R .

Рисунок 2.6 – Вилучення фрагментів з AST і оцінка їх подібності з програмою- прикладом (алгоритм 5)

Алгоритм 5 відвідує кожну вершину v AST G_t і аналізує кортеж послідовно виконуваних інструкцій ξ , пов'язаний з вершиною v (алгоритм 5). Так, в випадку, якщо вершина v є циклом, кортеж ξ містить все інструкції, послідовно виконувані на кожною ітерації циклу. У випадку, якщо v є функцією, кортеж ξ містить все інструкції, що послідовно виконуються при передачі управління у функцію. Якщо кортеж послідовно виконуваних інструкцій ξ не порожній, з нього витягуються все можливі комбінації з ϑ вершин AST $(v_1, v_2, \dots, v_P)$. Параметр ϑ алгоритма 14 пропонується встановити рівним числу вершин у кортежі послідовно виконуваних інструкцій, пов'язаних з коренем AST G_q.

Загальне число вилучених з ξ комбінацій при цьому оцінюється як:

$$C_{|\xi|}^{\vartheta} = \binom{|\xi|}{\vartheta} = \frac{|\xi|!}{\vartheta!(|\xi|-\vartheta)!}, \quad (2.4)$$

де ϑ – число вершин в вилучених з кортежу ξ комбінаціях, $|\xi|$ - довжина кортежу.

Для кожної видобутої комбінації (1, v_2 , … , v_P) створюється тимчасове AST G, що містить кортеж послідовно виконуваних інструкцій (1, v_2 , … , v_P), пов'язаних з вершиною v , таке AST у цій роботі називається фрагментом AST Gt. Алгоритм 5 будує множину R , що складається з пар (ρ, G), де G – фрагмент AST Gt , а ρ – відстань між фрагментом G і програмою-прикладом Gq . Результатом роботи алгоритму 5 є підмножина з пар, що входять з найменшими відстанями.

Оцінка якості алгоритма 5 з різними ваговими коефіцієнтами ω_1 , ω_2 і ω_3 проводилася на низці завдань (G_t, G_q, A) пошуку фрагментів у AST G_t за програмою-прикладом G_q . Відомості про розглянуті завдання пошуку фрагментів у текстах програм П$_1$, П$_2$, П$_3$ представлені у таблиці 2.2.

Таблиця 2.2 – Відомості о розглянутих завданнях

Завдання (G_t , G_q , A)	П$_1$	П$_2$	П$_3$
Число рядків коду в G_t	28	45	157
Число типів вершин AST в G_t	19	23	41
Усього фрагментів в AST G_t , $\|R\|$	146	116	289
Правильних фрагментів, $\|A\|$	4	3	5
Неправильних фрагментів, $\|R\| - \|A\|$	142	113	284
Значення параметра κ , $\|A\| + 2$	6	5	7

Програми G_t , характеристики яких наведені у таблиці 2.2, написані мовою Python, побудова AST здійснювалася засобами стандартної бібліотеки [6]. Програма G_t у завданні П$_2$ містить реалізацію логістичної регресії для завдань бінарної класифікації.

G_t у завданні П3 містить фрагмент веб-програми системи ЦАВ [5] . в AST G_t по програмі-прикладу G_q порівнювалися з безліччю правильних відповідей A з використанням заходів (2.1), (2.2). Параметр $к$ алгоритму 5 був встановлений рівним $| + 2$. результатів пошуку залежить від значень вагових коефіцієнтів ω_1 , ω_2 і ω_3 алгоритма 13, який використовується в алгоритмі 5. При цьому ω_1 визначає величину впливу ваг ребер ланцюгів Маркова для AST на відстань між програмами, що обчислюється за формулою (2.1), ω_2 визначає величину впливу ваг ребер ланцюгів Маркова для AST та для DU-графа, ω_3 визначає величину впливу міри подібності програм з урахуванням порівняння їх цикломатичної складності. У таблиці 2.3 наведено результати чисельної оцінки якості роботи алгоритма 5 при різних значеннях параметрів ω_1 та ω_2 .

Таблиця 2.3 – Результати чисельною оцінки алгоритма 5 в розглянутих

Задача	Ланцюг Маркова	ω_1	ω_2	Точность (3.1)	Полнота (3.2)	F_1-мера (3.3)
$П_1$	AST	1	0	0.17	0.25	0.20
	DU-граф	0	1	0.50	0.75	0.60
	AST и DU-граф	1	1	0.67	1.00	0.80
$П_2$	AST	1	0	0.57	0.80	0.67
	DU-граф	0	1	0.00	0.00	0.00
	AST и DU-граф	1	1	0.71	1.00	0.83
$П_3$	AST	1	0	0.20	0.33	0.25
	DU-граф	0	1	0.60	1.00	0.75
	AST и DU-граф	1	1	0.60	1.00	0.75

«AST» в таблиці 2.3 позначає, що при заданих значеннях параметрів ω_1 і ω 2 при порівняння програм за допомогою відстані Йенсена-Шеннона (алгоритм 4) використовується ланцюг Маркова тільки для AST (рисунок 2.2а), так як вага ω_2 дорівнює 0. "DU-граф" в таблиці 2.4 позначає, що при заданих значеннях параметрів ω_1 і ω_2 при порівнянні програм використовується ланцюг Маркова, побудова якого проводиться тільки для DU-графа (сині стрілки та вершини на рисунок 2.2б), оскільки вага ω_1 дорівнює 0. "AST і DU-граф" у таблиці 3.4 позначає, що при заданих значеннях ω 1 і ω 2 використовується ланцюг Маркова, побудова якої виробляється як для AST, так і для DU-графа (Див. чорні і сині стрілки і вершини на рисунок 2.2б). Жирним в таблиці 2.4 виділено найкращі результати оцінки якості в конкретної задачі і найкраща модель. Як показано в таблиці 2.4, векторні уявлення текстів програм на основі ланцюгів Маркова для AST та для DU-графа дозволяють підвищити точність (2.1), повноту (2.2) та F 1 -міру (2.3) у розглянутій задачі пошуку фрагментів в AST, по порівнянні з векторними уявленнями текстів програм на основі ланцюгів Маркова тільки для AST або тільки для DU-графа, в середньому на 27%. При цьому, якість пошуку фрагментів текстів програм в сенсі (2.1), (2.2) і (2.3) для різних значень параметрів ω 1 та ω 2 залежить від завдання, що розглядається.

Для візуальної оцінки якості векторних уявлень фрагментів AST на основі ланцюгів Маркова, побудова яких виробляється для AST, і векторних уявлень фрагментів AST на основі ланцюгів Маркова, побудова яких виробляється як для AST, так і для DU-графів, було виконано зниження розмірності з метою вкладення векторних уявлень у простір $\mathbb{R}^2$, придатне для візуалізації. Для зниження розмірності простору використовувалася модифікація алгоритму UMAP [7] та метод оптимізації на основі градієнтного спуску з адаптивним кроком Adam [10]. Алгоритм UMAP виконує зниження розмірності векторів, що належать простору високої розмірності, на основі попередньої побудови графа подібностей найближчих сусідів кожного вектора [5]. Отримані при допомоги UMAP візуалізації 50 найбільше схожих в сенсі ρ

(див. рядок 7 в Алгоритм 13) на G_q фрагментів AST G_t при $\omega_1 = 1$ $\omega_2 = 1$ і $\omega_3 = 1$ для задач $П_1$, $П_2$, $П_3$ представлені на рисунку 2.7.

Круглі червоні крапки на рисунку 2.7 показані розміщені алгоритмом UMAP у просторі R^2 такі векторні уявлення фрагментів AST G_t, які входять в безліч правильних відповідей A завдання (G_t, G_q, A) і в насправді відповідні програмі-прикладу G_q. У вигляді зеленого ромба на рисунок 2.7 показано векторне подання програми-прикладу G_q, також розміщене алгоритмом UMAP в просторі R^2. У вигляді синіх хрестів на рисунку 2.7 показано векторні уявлення фрагментів AST G_t, які не входять до множини правильних відповідей A.

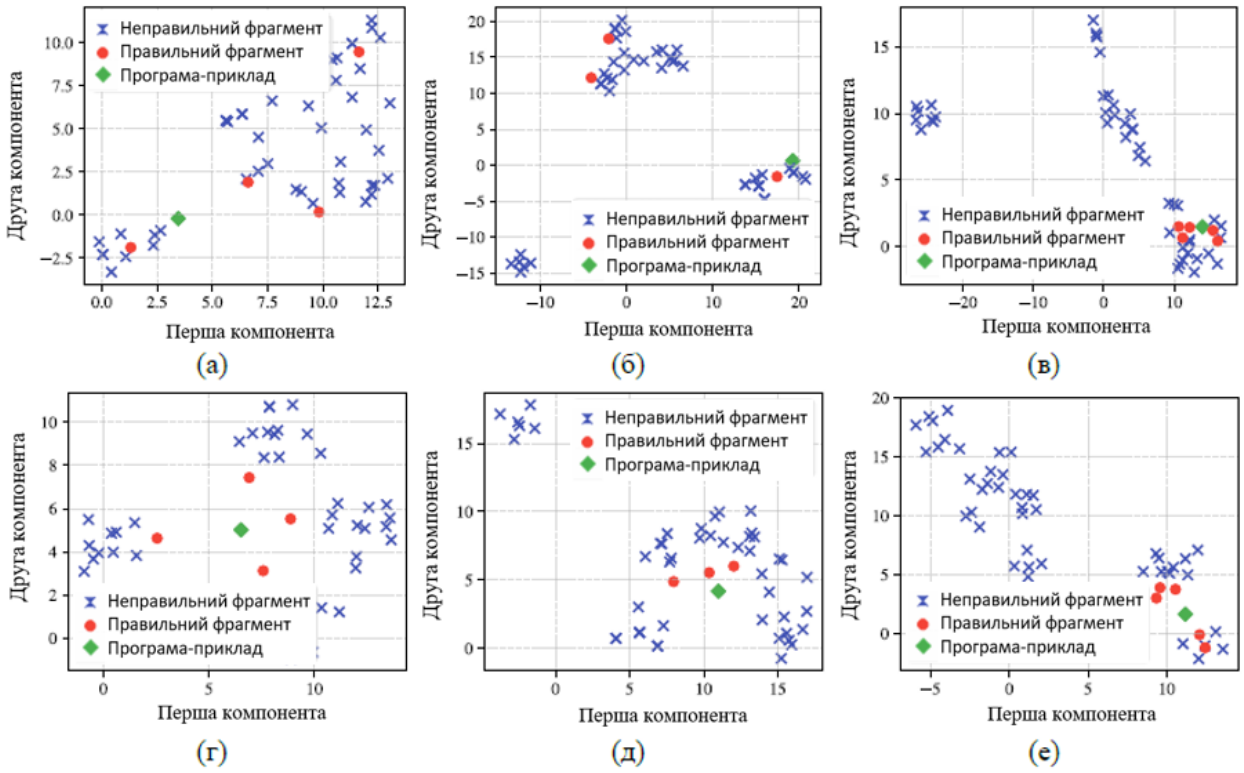


Рисунок 2.7 – Візуалізації векторних уявлень фрагментів AST G_t на основі ланцюгів Маркова для AST після їх вкладення в простір R^2

Як показано на рисунку 2.7, векторні уявлення фрагментів AST G_t, насправді відповідні програмі-прикладу G_q, розміщені алгоритмом UMAP в просторі низькою розмірності ближче до вектора програми-прикладу G_q в

порівнянні з фрагментами, що не входять до множини відповідей A , в випадку використання ланцюгів Маркова для AST і для DU-графа (рисунок 2.7г, д, е і рисунок 2.2б) по порівнянні з ланцюгами Маркова, побудова яких виробляється тільки для AST (рисунок 2.7а, б, рисунок 2.2а), що особливо помітно на рисунок 2.7г, д в порівнянні з рисунок 2.2а, б.

Для вибору оптимальних значень вагових коефіцієнтів ω_1 , ω_2 і ω_3 в алгоритмі 4 в кожною з розглянутих завдань (G_t , G_q , A) , відомості про які наведені в таблиці 2.3, вирішувалося завдання мінімізації безперервної функції оцінки втрат за допомогою алгоритму 4. Вибрані значення вагових коефіцієнтів ω_1 , ω_2 , ω_3 для кожною з завдань, а також відповідні їм значення наведено у таблиці 2.5.

2.5 Результати застосування методики винесення рекомендацій по прискорення програм

Методика винесення рекомендацій щодо підвищення швидкості виконання програм, що включає 6 кроків і заснована на спільному використанні алгоритму пошуку фрагментів AST по програмі-прикладу, алгоритму обчислення відстані

між векторними уявленнями текстів програм на основі ланцюгів Маркова, побудова яких виробляється як для AST, так і для DU-графа, а також на основі порівняння цикломатичної складності програм (алгоритм 3), використовувалася для винесення рекомендацій щодо перетворення текстів програм на мовою Python, реалізують: алгоритм навчання логістичної регресії для задач бінарної класифікації;

Аналіз текстів програм по запропонованої методиці виконувався з метою їх прискорення шляхом переміщення фрагментів текстів програм на спеціалізовані прискорювачі, доступні на заданій гетерогенної обчислювальної платформі, характеристики якої наведено в таблиці 2.4.

Таблиця 2.4 – Характеристики заданою гетерогенної обчислювальної платформи

Параметр	Значення
CPU	Intel® Core™ i7-4770, 3.40 GHz (4 ядра)
GPU	NVIDIA GeForce GTX 1050TI, 1341 MHz, 4 GB GDDR5
Оперативна пам'ять	16 GB DDR3 (1600 MHz)

На першому кроці методики було сформовано базу реалізацій простих алгоритмів мовою Python, таких як алгоритм обчислення скалярного твори векторів, алгоритм транспонування матриці, алгоритм застосування математичних операцій до кожного елемента вектора.

На другому кроку методики кожна з реалізацій алгоритмів на мовою Python транслювалася в уявлення, придатні для запуску на кожному з прискорювачів, доступних на заданій гетерогенний обчислювальної платформі (таблиця 2.4). Для розпаралелювання обчислень на CPU і GPU за принципом SIMD (Single Instruction, Multiple Data) використовувався DSL PyTorch [4].

на третьому кроку методики оцінювалося час роботи програмних реалізацій алгоритмів на кожному з прискорювачів, доступних на заданій гетерогенній обчислювальній платформі (таблиця 2.6). При оцінки швидкості виконання на вхід програмним реалізаціям алгоритмів подавались випадкові вектори, належать простору R^{10^6}, обчислення повторювалися 2000 раз, після чого обчислювалося середнє значення часу роботи фрагмента на прискорювачі.

на четвертому кроку методики база програмних реалізацій алгоритмів доповнювалась коефіцієнтами, обчисленими на основі оцінок швидкості виконання алгоритмів на прискорювачах, доступних на гетерогенній обчислювальній платформі. Сформована таким чином база програмних реалізацій алгоритмів на мовою Python представлена в таблиці 2.5. Жирним в

таблиці 2.5 виділено коефіцієнти прискорення, великі одиниці. Як видно з таблиці 2.5, обчислювально-інтенсивні операції з векторами, що належать R^{10^6}, на заданій гетерогенній обчислювальній платформі, характеристики якої представлені в таблиці 3.6, вигідніше виконувати на GPU. У випадку скалярного твори векторів GPU виявляється швидше CPU в 2.31 рази, а у випадку застосування арифметичних операцій до вектору GPU виявляється швидше CPU в 25.13 разів. Транспонування матриць на заданій гетерогенній обчислювальній платформі (таблиця 2.6) краще виконувати на CPU.

Таблиця 2.6 – База програмних реалізацій алгоритмів з коефіцієнтами прискорення

Програма-приклад G_q	SIMD на CPU	SIMD на GPU	Коеф. CPU	Коеф. GPU
out = 0 for i in range(len(a)): out = out + b[i] * a[i]	torch.dot(a, b)	torch.dot(a.to('cuda'), b.to('cuda'))	0.43	2.31

transpose = []
for i in range(len(m[0])):
t = []
for j in range(len(m)):
t.append(m[j][i])
transpose.append(t)
	torch.transpose(m, 0, 1)	torch.transpose(m.to('cuda'), 0, 1)	1.41	0.71

out = [] for i in range(len(a)): out.append((a[i]) + a[i]) / a[i])	torch.divide(torch.add(a, a), a)	b = a.to('cuda') torch.divide(torch.add(b, b), b)	0.04	25.13

На основі алгоритма 3 було розроблено розширення для редактора коду Visual Studio Code® з підтримкою пошуку фрагментів текстів програм, що

потенційно придатні для прискорення (таблиця 2.7), показу кращого прискорювача для розпаралелювання програми по принципу SIMD, показу коефіцієнтів поліпшення швидкості виконання при перенесенні обчислень на попередню апаратний прискорювач по порівнянні з іншими доступними на заданою гетерогенної обчислювальної платформи апаратними прискорювачами (рисунок 2.7).

На п'ятому кроку методики винесення рекомендацій по прискорення програм при допомоги розробленого розширення для редактора Visual Studio Code® виконувався пошук фрагментів текстів програм по програм-прикладів G_q з сформованою бази програмних реалізацій алгоритмів (Див. таблицю 3.7). на шостому кроку методики формувався звіт з рекомендаціями по вищеню швидкості виконання програми для заданою гетерогенної обчислювальної платформи, формування звіту здійснювалося при допомоги API редактора Visual Studio Code®. Отриманий звіт подано на рисунок

Для оцінки якості рекомендацій, показаних на рисунок 2.7 було проведено розпаралелювання . проаналізованих по запропонованої методиці текстів програм, реалізують алгоритм навчання логістичної регресії для завдань бінарної класифікації; алгоритм виконання передбачення машиною екстремального навчання по формулі, попередньо навченої по формулі для завдань багатокласової класифікації. Для кожною програми розглядалися 3 варіанти розпаралелювання по принципом SIMD.

Studio Code®, отриманих на основі пошуку фрагментів у AST за програмою-прикладом для аналізованої програми: а) реалізуючою алгоритм навчання логістичної регресії для задач бінарної класифікації; б) реалізує алгоритм виконання передбачень машиною екстремального навчання за формулою для завдань багатокласової класифікації Фрагменти текстів програм, для яких найкращим прискорювачем є GPU, об'єднувалися і транслювалися в DSL PyTorch з підтримкою CUDA. Фрагменти текстів програм, для яких кращим прискорювачем є CPU, транслювалися у DSL PyTorch з підтримкою паралелізму в CPU. на вхід програмі, реалізуючою

алгоритм навчання логістичної регресії для завдань бінарної класифікації, подавалися вектори, що належать простору R^{10} , із синтетичного набору векторів для бінарної класифікації, згенерованого за допомогою бібліотеки sklearn [101], було виконано 200 ітерацій навчання алгоритму.

```
def logistic_regression(alpha, x, y, iterations):
    Предпочтительно распараллеливание на CPU (в 1.41 раз быстрее чем на GPU)
    transpose = []
    theta = [0] * len(x[0])
    for i in range(len(x[0])):
        features = []
        for j in range(len(x)):
            features.append(x[j][i])
        transpose.append(features)
    for _ in range(iterations):
        outputs = []
        for example in x:
            Предпочтительно распараллеливание на GPU (в 2.31 раз быстрее чем на CPU)
            output = 0
            for i in range(len(example)):
                output = output + example[i] * theta[i]
            outputs.append(output)
        Предпочтительно распараллеливание на GPU (в 25.13 раз быстрее чем на CPU)
        activations = []
        for i in range(len(outputs)):
            activations.append(1 / (1 + exp(1 - outputs[i])))
        Предпочтительно распараллеливание на GPU (в 25.13 раз быстрее чем на CPU)
        multipliers = []
        for i in range(len(y)):
            multipliers.append((activations[i] - y[i]) / len(y))
        gradient = []
        for features in transpose:
            Предпочтительно распараллеливание на GPU (в 2.31 раз быстрее чем на CPU)
            component = 0
            for i in range(len(x)):
                component = component + features[i] * multipliers[i]
```

```
def elm_predict(samples, alpha, bvec, beta):
    Предпочтительно распараллеливание на CPU (в 1.41 раз быстрее чем на GPU)
    alphatranspose = []
    for i in range(len(alpha[0])):
        row = []
        for j in range(len(alpha)):
            row.append(alpha[j][i])
        alphatranspose.append(row)
    hidden = []
    for i in range(len(samples)):
        row = []
        for j in range(len(alphatranspose[0])):
            Предпочтительно распараллеливание на GPU (в 2.31 раз быстрее чем на CPU)
            cell = 0
            for k in range(len(samples[0])):
                cell = cell + samples[i][k] * alphatranspose[k][j]
            row.append(cell)
        hidden.append(row)
    bias = []
    for hiddenrow in hidden:
        Предпочтительно распараллеливание на GPU (в 25.13 раз быстрее чем на CPU)
        row = []
        for j in range(len(hiddenrow)):
            row.append(hiddenrow[j] + bvec[j])
        bias.append(row)
    activated = []
    for biasrow in bias:
        Предпочтительно распараллеливание на GPU (в 25.13 раз быстрее чем на CPU)
        row = []
        for j in range(len(biasrow)):
            row.append(1 / (1 + exp(-biasrow[j])))
```

(а) (б)

Рисунок 2.8 — Візуалізація результатів статичного аналізу текстів програм у редакторі Visual

На вхід програмі, що реалізує алгоритм виконання передбачень машиною екстремального навчання за формулою, подавався набір із 10^4 векторів, належать простору R^{10^4} , число нейронів прихованого шару було встановлено рівним 10^3 . Виміри часу роботи повторно виконували 50 разів для кожного з трьох варіантів розпаралелювання кожної з розглянутих програм, результати наведені в таблиці 2.7.

В таблиці 2.7 виділено найкраще значення: часу навчання програмного реалізації логістичної регресії для завдань бінарної класифікації; часу виконання передбачень програмною реалізацією машини екстремального навчання для завдань багатокласовий клас -сифікації.

Встановлено, що внаслідок дотримання рекомендацій щодо переробки тексту програми, сформованих по запропонованої методиці винесення рекомендацій по підвищення швидкості виконання програми, що реалізує алгоритм навчання логістичної регресії для завдань бінарної класифікації

(рисунок 2.8а), її швидкість виконання на заданою гетерогенної обчислювальної платформі, характеристики якої наведено в таблиці 2.6, виявляється найкращою серед розглянутих варіантів. Швидкість виконання програми покращується у 5.23 рази порівняно з CPU, у 1.04 рази порівняно з GPU.

Таблиця 2.7 – Час роботи трьох варіантів розпаралелювання програм

Тип розпаралелювання	Тільки CPU	Тільки GPU	CPU+GPU за рекомендаціями
Час навчання логістичної регресії для завдань бінарної класифікації, с.	20.91	4.17	4.00
Час виконання передбачень машиною екстремального навчання задач багатокласової класифікації, с.	6.72	1.58	1.34

Встановлено, що внаслідок дотримання рекомендацій щодо переробки тексту програми, сформованих по запропонованої методиці винесення рекомендацій по підвищення швидкості виконання програми, що реалізує алгоритм виконання передбачень машиною екстремального навчання для завдань багатокласової класифікації (рисунок 2.7б), її швидкість виконання на заданій гетерогенної обчислювальної платформі, характеристики якої наведено в таблиці 2.7, виявляється найкращою серед розглянутих варіантів. Швидкість виконання програми покращується у 5.01 разів у порівнянні з CPU, у 1.18 разів у порівнянні з GPU.

3 МЕТОДИ І АЛГОРИТМИ ОБЛІКУ НАВЧАЛЬНИХ ДОСЯГНЕНЬ В КУРСАХ ПО ПРОГРАМУВАННЯ

3.1 Методика обліку навчальних досягнень в масових курсах по програмування

Цифровізація наводить до підвищеного попиту на спеціалістів у галузі інформаційних технологій, зокрема, на розробників програмного забезпечення, чим обумовлюється масовий характер сучасних університетських курсів по програмуванню. У результаті підвищується навантаження на педагогічних працівників. Для викладачів стає важко, а часом і неможливо, уважно поставитися до всіх студентів, придумати достатнє число оригінальних навчальних завдань по програмуванню в умовах, коли запозичення студентами чужих рішень стає нормою. Наслідком підвищеною навантаження на викладача може стати зниження його залучення до навчального процесу, зникнення бажання підходити творчо до процесу викладання. У цією зв'язку допомога може надати автоматизація навчання програмування. В даний час в освітній процес впроваджуються системи управління навчанням. Для запобігання запозичення студентами чужих рішень в таких системах використовуються алгоритми виявлення збігаються блоків коду, такі як алгоритми, засновані на попарних порівняннях наборів зважених ключових слів, вилучених із програм [8], алгоритми, засновані на порівнянні AST [9]. Однак, на практиці учні здатні уникнути спрацювання подібних алгоритмів виявлення запозичень в програмах, виконавши просту переробку коду, а хибні спрацьовування алгоритмів може призвести до відхилення коду, самостійно написаного студентом.

Одним з підходів для вирішення проблеми унікальних завдань по програмуванню різних типів, заснований на загальним методі рішення завдання задоволення обмеженням «згенерувати і перевірити» [5]. Цей підхід

використовується в системі ЦАВ, яка автоматизує породження та перевірку унікальних завдань для масового курсу програмування мовою Python. У курсі програмування мовою Python кожному студенту пропонується вирішити індивідуальний набір унікальних завдань різних типів, при цьому студент не обмежений у виборі способу вирішення кожної з завдань і може користуватися усіма інструментами, доступними в стандартною бібліотеці мови Python. Під способом рішення тут розуміється обраний студентом алгоритм рішення навчальної задачі. Оскільки різні способи вирішення однієї і тієї ж задачі мають і сильні, і слабкі сторони, для більш детального вивчення можливостей мови програмування корисно мотивувати студентів до розгляду різних способів рішення задач. Використання систем досягнень в освітній процес позитивно позначається як на часі, проведеному студентами за вирішенням завдань, так і на успішності, показано, що використання ігрових підходів в освітній процес підвищує залучення студентів.

Для мотивації студентів до вирішення унікальних завдань із програмування різними способами пропонується методика автоматизованого обліку навчальних досягнень, заснована на застосуванні Алгоритма 2 і Алгоритма 3 для перетворення програм в векторні представлення на основі ланцюгів Маркова для AST, а також алгоритмів кластеризації та класифікації векторних уявлень програм. Схема методики обліку навчальних досягнень в масових курсах з програмування, де застосовується LMS із підтримкою автоматичного породження та перевірки завдань, представлена на рисунок

Запропонована методика автоматизованого обліку навчальних досягнень у масових курсах програмування, в яких застосовуються LMS з підтримкою автоматичного породження та перевірки унікальних завдань, що включає наступні кроки [6].

Крок 1. Формування набору, що складається з перевірених і прийнятих LMS текстів програм, які вирішують унікальні автоматично згенеровані завдання заданого типу.

Крок 2. Перетворення програм із сформованого набору у векторні

уявлення на основі ланцюгів Маркова, побудова яких виробляється для AST, по Алгоритму 9, з подальшим збереженням безлічі H , містить типи зустрічаються в програмах з набору вершин AST.

Крок 3. Розбиття векторних уявлень програм на кластери агломеративним алгоритмом ієрархічної кластеризації по методом середньої зв'язку для виявлення способів вирішення завдань, вибір числа кластерів здійснюється по найбільшому значенням індексу кластерного силуету.

Крок 4. Фільтрація виявлених кластерів, що відповідають способам рішення, на основі порогового значення числа програм, що входять до кластера, що визначається вручну.

Крок 5. Розробка класифікаторів векторних уявлень текстів програм на основі ланцюгів Маркова для автоматичного визначення способу рішення завдання по векторному Представленню програми як мітки класів використовуються мітки кластерів, отримані на кроці 3, після їх фільтрації на кроці 4.

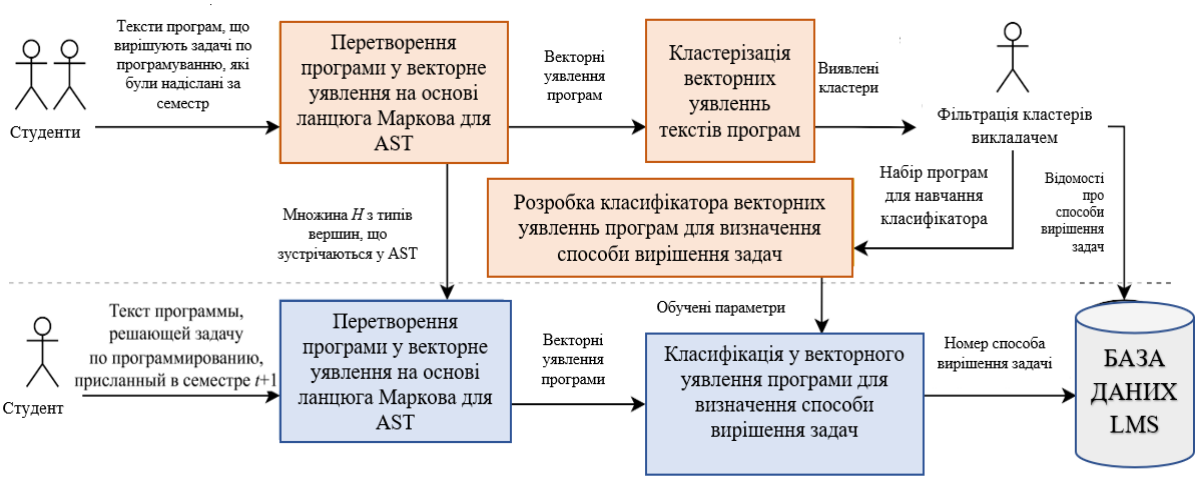


Рисунок 3.1 – Схема методики обліку навчальних досягнень у масових курсах з програмування, де застосовується LMS з підтримкою породження і перевірки завдань

Крок 6. Інтеграція розроблених класифікаторів векторних подань програм у LMS для автоматичного обліку навчальних досягнень, причому

перетворення програм на векторні подання виконується за Алгоритмом 15 на основі множини H , отриманої на кроку 2 в результаті роботи алгоритму 3, а розв'язання задачі у додатковий спосіб вважається навчальним досягненням. Відомості про досягнення зберігаються в базі даних LMS (рисунок 3.1) і показуються в інтерфейсі користувача, який бачать студенти та викладачі.

Вхід: G D AST програми для перетворення в векторне уявлення.
H D Безліч типів вершин AST.
$n \in$ N D Порядок ланцюги Маркова для AST, $n \geq 1$.

1. $G_M \leftarrow$ **Алгоритм 3** (G, n). D Побудувати ланцюг Маркова $G_M = (P, M)$ порядку n .
2. $h = |H|$. D Число типів вершин AST, зустрічаються в ланцюгах Маркова.
3. **Перетворити** матрицю суміжності $\mathbf{p} \in \mathrm{R}^{h \times h}$ графа $G_* = (H, M)$ до $\vec{v} \in \mathrm{R}^m$, $m = h^2$.
4. **Повернути** векторне подання програми $\vec{v} \in \mathrm{R}^m$.

Рисунок 3.2 – Перетворення AST в векторне подання на основі ланцюги Маркова (алгоритм 6)

У запропонованій методиці передбачається, що LMS вже використовувалася для автоматизації породження і перевірки завдань по програмування в семестрі з номером t . на основі надісланих студентами в LMS програм формується набір перевірених і прийнятих рішень унікальних завдань заданого типу з метою їх наступною кластеризації і підготовки класифікаторів для обліку навчальних досягнень у семестрі t +1 [4, 5]. Таким чином, у семестрі t +1 облік навчальних досягнень здійснюється автоматично, на основі даних, зібраних за семестр t . Крім того, результати кластеризації програм можуть використовуватися викладачами для ідентифікації пробілів в навичках і знаннях студентів, а також для оцінки якості генераторів унікальних завдань з програмування. Генератор завдань вважається незадовільним, якщо завдання можуть бути вирішені малим числом способів [7].

Допоміжний алгоритм 5 дозволяє виконувати перетворення програм у векторні уявлення в семестрі t +1 на основі множини H, обчисленого в

результаті застосування Алгоритму 9 до програм, надісланих в LMS за семестр t .

3.2 Виявлення способів рішення завдань на основі кластеризації програм

Запропонована методика автоматизованого обліку навчальних досягнень у масових курсах програмування впроваджено в LMS ЦАВ [153], автоматизуючу ряд видів діяльності викладача курсу «Програмування на мовою Python» і підтримуючу автоматичне породження і перевірку унікальних завдань 11 різних типів [15]. Список типів завдань складається з ядра та веб-додатків. Ядро LMS забезпечує породження умов унікальних завдань 11 типів для кожного студента, наборів відкритих і закритих тестів для кожною завдання [15], а також здійснює перевірку що надсилаються рішень. Перевірка рішень в LMS складається з наступних етапів.

Перший етап – це етап статичного аналізу текстів програм, що включає перевірку коду на відповідність стандарту оформлення коду PEP8, а також оцінку когнітивний складності текстів програм.

Другий етап – це етап тестування програми, причому тестування проводиться у «пісочниці», представленій у LMS ЦАВ у вигляді Docker-контейнера із середовищем виконання gVisor [6], використовується з метою захисту сервера LMS ЦАВ від несанкціонованого доступу. Веб-додаток LMS ЦАВ забезпечує можливість людино-машинного взаємодії з ядром, надаючи інтерфейс користувача для перегляду списку груп, перегляду статусу виконання завдань студентами обраною групи, відправки рішення в LMS ЦАВ на перевірку, перегляд статусу завдання.

Для оцінки якості програм в LMS ЦАВ використовується метрика когнітивний складності, причому програми з перевищеною когнітивний складністю автоматично відхиляються LMS ЦАВ на етапі їх статичного аналізу при пороговому значенні когнітивної складності, рівному 10. У відмінність від цикломатичної складності [6], в когнітивний складності [4]

додаються додаткові штрафи за вкладені синтаксичні конструкції, що приводять до розгалуження потоку управління програми. Множина R_1 , R_2 та R_3 у реалізації алгоритму 3, що використовується в LMS, формуються для абстрактного синтаксису мови програмування Python, специфікація якого наведена в [9]. Правила з множина R_1 , що використовуються LMS для оцінки когнітивної складності програм мовою Python, наведено у таблиці 3.1.

Таблиця 3.1 – Множина правил R_1 для оцінки когнітивний складності програм на мовою Python

№	Опис правила	Відображення r: $V \times V \to$ B	Відображення w: $V \to$ Z
1	Умовний оператор з 2 і більше розгалуженнями	$\tau(v)$ = If $\wedge$ | v. orelse | > 0 $\wedge$ $\neg$ (| v . orelse | = 1 $\wedge$ $\tau(v.\text{orelse}_0)$ = If) .	1.
2	Рекурсія	$\tau(v) \in$ { FunctionDef, AsyncFunctionDef } $\wedge$ $\exists v_c \in V_v$: ($\tau(v_c)$ = Call $\wedge$ $\tau(v_c.$ func) = Name $\wedge$ v_c . func. id = v. name).	1.
3	Розгалуження в циклах та обробниках винятків	$\tau(v) \in$ { For, AsyncFor, While, Try } $\wedge$ | v. orelse | > 0 .	1
4	Конструкції, що приводять до розгалуження потоку управління програми	$\tau(v) =$ { ListComp, DictComp, SetComp, GeneratorExp, For, AsyncFor, While, Match, ExceptHandler, IfExp, If }.	1.
5	Генератори і розгалуження у генераторах	$\tau(v) \in$ { ListComp, DictComp, SetComp, GeneratorExp }.	$\sum_v g \in G$ |v_g . ifs| , де $G = v$. generators .
6	Логічні оператори	$\tau(v)$ = BoolOp .	1

У таблиці 3.1 τ: $V \to T$ позначає відображення, яке ставить в відповідність вершині $v \in V$ її тип $t \in T$, де T – Множина можливих типів

вершин, перелічене [9] для абстрактного синтаксису мови Python; $r: V \times V \to \mathrm{B}$ – відображення, що приймає на вхід пару (v, v_p) , що складається – з вершини AST$v \in V$ і вершини $v_p \in V$, що є батьківської по відношенню до v; $w: V \to \mathrm{Z}$ – відображення, обчислювальне величину, додається до складності, не залежить від рівня вологості. Як показано в таблиці 3.1, містить правила з множини R_1 , що є параметром fлгоритма 3, значення когнітивний складності програми збільшується для кожного умовного оператора з 2 і більше розгалуженнями, для кожною рекурсивний функції, для розгалужень в циклах, в обробниках винятків та в генераторах (таблиця 3.1), для ланцюжків логічних операторів. Правила, перелічені в таблиці 3.1, відповідають безумовним правилам збільшення когнітивної складності програми, сформульованої SonarSource™.

Правила з множини R_2 алгоритма 3, використовувані LMS ЦАВ при оцінки складності програм мовою Python, наведені в таблиці 3.2.

Таблиця 3.2 – Множини правил R_2 для оцінки когнітивний складності програм мовою Python,

№.	Опис правила	Відображення $r: V \times V \to \mathrm{B}$	Відображення $w: V \to \mathrm{Z}$
1.	Умовний оператор, не є else-if	$\tau(v) = \mathrm{If} \wedge \neg(\tau(v_p) = \mathrm{If} \wedge \lvert v_p.\mathrm{orelse}\rvert = 1 \wedge v_p.\mathrm{orelse}_0 = v)$.	1
2.	Функція, не яка є декоратором	$\tau(v) \in F \wedge \neg(\lvert v.\mathrm{body}\rvert = 2 \wedge \tau(v.\mathrm{body}_0) \in F \wedge \tau(v.\mathrm{body}_1) = \mathrm{Return})$, де F ={ FunctionDef, AsyncFunctionDef } .	1
3.	Синтаксис, що збільшує вкладеність	ListComp, DictComp, SetComp, GeneratorExp, For, AsyncFor, While, $\tau(v) \in$	1

У таблиці 3.2 $\tau: V \to T$ позначає відображення, ставить в відповідність вершині $v \in V$ її тип $t \in T$; $r: V \times V \to \mathrm{B}$ – відображення, приймаюче на вхід пару (v, v_p) , що складається з вершини AST $v \in V$ і вершини $v_p \in V$, яка є

батьківською по відношенню до v ; w: $V \to$ Z – відображення, яке обчислює величину, яка додається до поточного рівня вкладеності λ. Правила з множини R_3, використовувані LMS ЦАВ для оцінки складності програм мовою Python, наведено у таблиці 3.3.

Таблиця 3.3 – Множина правил R 3 для оцінки когнітивний складності програм мовою Python,

№.	Опис правила	Відображення r: $V \times V \to$ B	Відображення w: $V \to$ Z
1.	Умовний оператор, не є else-if	τ (v) = If $\wedge$ $\neg(\tau(v_p)$ = If $\wedge$ $\lvert v_p$. orelse $\rvert$ = 1 $\wedge$ v_p . orelse $_0$ = v) .	1.
2.	Конструкції, що приводять до розгалуження потоку управління програми	ListComp, DictComp, SetComp, τ (v) $\in$ {GeneratorExp, For, AsyncFor}. While, Match, ExceptHandler, IfExp	1

У таблиці 3.3 τ: $V \to T$ позначає відображення, ставить в відповідність вершині $v \in V$ її тип $t \in T$; r: $V \times V \to$ B - відображення, приймаюче на вхід пару (v, v_p) , що складається з вершини AST $v \in V$ і вершини $v_p \in V$, яка є батьківською по відношенню до v ; w: $V \to$ Z – відображення, яке обчислює величину, що додається до складності, яка залежить від поточного рівня вкладеності λ.

Когнітивна складність програм оцінюється LMS ЦАВ правил з множин R_1 , R_2 і R_3 , елементи яких перераховані в таблиці 3.1, 3.2 і 3.3 відповідно.

За весняний семестр 2021-2022 навчального року до LMS ЦАВ було надіслано 66 553 програми, що вирішують унікальні автоматично згенеровані завдання, з них 52 483 програми були автоматично відхилені через порушення стандарту форматування PEP8, перевищення когнітивної складності програми при пороговому значення, рівному 10, синтаксичних помилок або некоректної роботи, 14 070 програм було прийнято. Порівняння числа прийнятих та відхилених LMS ЦАD програм, що вирішують унікальні завдання 11 різних

типів, представлено на рисунку 3.3.

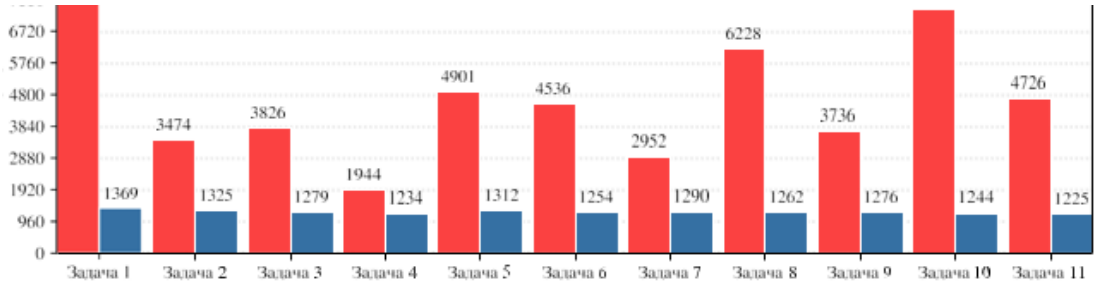


Рисунок 3.3 – Порівняння числа прийнятих і відхилених LMS ЦАВ текстів програм, вирішальних унікальні завдання 11 різних типів

Як видно із рисунку 3.3, кількість прийнятих програм залежить від типу завдання, при цьому перші завдання характеризуються найбільшим числом прийнятих рішень. Для кожного типу завдання значне число програм було відхилено. Методика автоматизованого обліку навчальних досягнень застосовувалася тільки до наборам перевірених і прийнятих системою LMS ЦАВ текстів програм, окремо для кожного з 11 наборів текстів програм, вирішальних унікальні завдання заданого типу. на першим кроку методики формувалися 11 наборів текстів програм, містять прийняті LMS ЦАВ рішення автоматично згенерованих завдань 11 типів (рисунок 3.3). на другому кроку методики набори текстів програм перетворювалися в набори їх векторних уявлень за алгоритмом 1. Для цього спочатку виконувалася побудова AST для кожної програми за допомогою засобів, доступних у стандартній бібліотеці мови Python 3.10. Потім в отриманому наборі AST вироблялася заміна ряду типів вершин AST, таблиця 3.4 визначає список замінних типів вершин і типів-замінників. Заміни типів вершин AST проводились для зменшення відмінностей між програмами, що використовують різні синтаксичні конструкції і структури даних для реалізації схожих або одних і тих самих алгоритмів.

Виходом алгоритма 1 є Векторний набір B уявлень текстів програм, а також множина H , містить типи вершин, зустрічаються в AST програм. У таблиці 3.5 наведено характеристики оброблених наборів текстів програм. Як

видно з таблиці 3.5, число компонент m в векторних уявленнях текстів програм з наборів, відповідних різним типом завдань, має тенденцію до зростання зі збільшенням номера задачі.

Таблиця 3.4 – Заміни типів вершин AST перед застосуванням алгоритма 1

Заміни для завдань з 1 по 11		Заміни для завдань з 6 по 11	
Замінюване	Замінник	Замінюване	Замінник
Constant, Attribute	Name	List, Tuple, Set	List
BoolOp, Call, UnaryOp, BinOp	Op	Lambda, JoinedStr, FormattedValue	Name
Lt, LtE	Less		
Gt, GtE	Greater	Pass, Break, Continue	-
AugAssign, AnAssign	Assign	ExceptHandler	Try
match_case, MatchStar, MatchAs, MatchOr, MatchSingleton, MatchSequence, MatchMapping, MatchClass	MatchValue	For, While, ListComp, SetComp, DictComp, GeneratorExp, comprehension	Loop
UAdd	Add	IfExp	If
USub	Sub		

Таблиця 3.5 – Характеристики оброблених наборів текстів програм

Тип завдання	1	2	3	4	5	6	7	8	9	10	11
Розмір набору	1369	1325	1279	1234	1312	1254	1290	1262	1276	1244	1225
Потужність H	22	30	28	31	28	32	28	34	29	36	34
Число компонент m	484	900	784	961	784	1024	784	1156	841	1296	1156

На третьому кроці методики виконувалося розбиття набору векторних уявлень програм на кластери агломеративним алгоритмом ієрархічної кластеризації за методом середнього зв'язку [7] для виявлення способів розв'язання задач, використовувалася реалізація алгоритму з бібліотеки sklearn [10]. Вибір числа кластерів здійснювався з урахуванням значень індексу кластерного силуету з діапазону від 15 до 30 кластерів, у своїй індекс був максимізовано для кожного набору даних. Графічні залежності індексу кластерного силуету від числа кластерів для завдань різних типів представлені на рисунку 3.4.

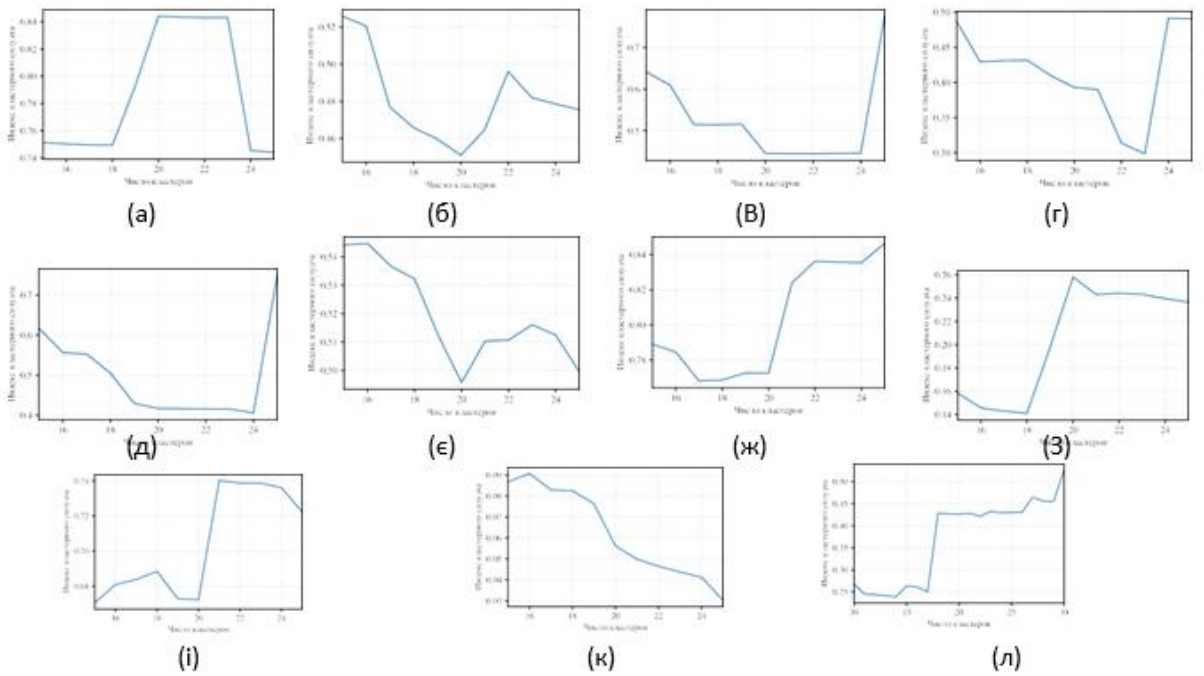


Рисунок 3.4 — Графічні залежності індексу кластерного силуету від числа кластерів до завдань різних типів: а) завдання 1; б) завдання 2; в) завдання 3; г) Завдання 4; д) завдання 5; є) завдання 6; ж) завдання 7; з) Завдання 8; і) завдання 9; к) завдання 10; л) завдання 11

На четвертому кроці методики виконувалася фільтрація числа кластерів на основі порогового значення мінімально допустимого числа програм, що входять до кластера, порогові значення для кожного набору програм

визначалися викладачем вручну. У таблиці 3.6 для кожного набору текстів програм наведено найбільше значення індексу кластерного силуету, обране з його основі число кластерів (рисунок 3.4), граничне значення мінімально допустимого числа програм, які входять у кластер, і навіть число кластерів після фільтрації.

Таблиця 3.6 – Найбільше значення індексу кластерного силуету

Тип завдання	1	2	3	4	5	6	7	8	9	10	11
Значення індексу (1.45)	0.84	0.53	0.78	0.49	0.75	0.55	0.85	0.23	0.74	0.10	0.53
Усього кластерів	20	15	25	24	25	16	25	20	21	16	30
мін. програм в кластері	10	10	10	10	10	30	10	10	30	10	70
Кількість кластерів	4	4	3	5	5	8	6	5	8	5	5

Виявлене число кластерів після фільтрації для завдань 11 типів представлено на рисунок 3.4. Синім рисунку 3.4 позначені типи завдань, котрим здійснюються базові заміни AST вузлів при побудові ланцюгів Маркова, червоним позначені типи завдань, для яких здійснюються розширені заміни: такі типи вершин AST як For і While замінюються Loop, такі типи вершин як List, Tuple, Set замінюються на List (таблиця 3.4).

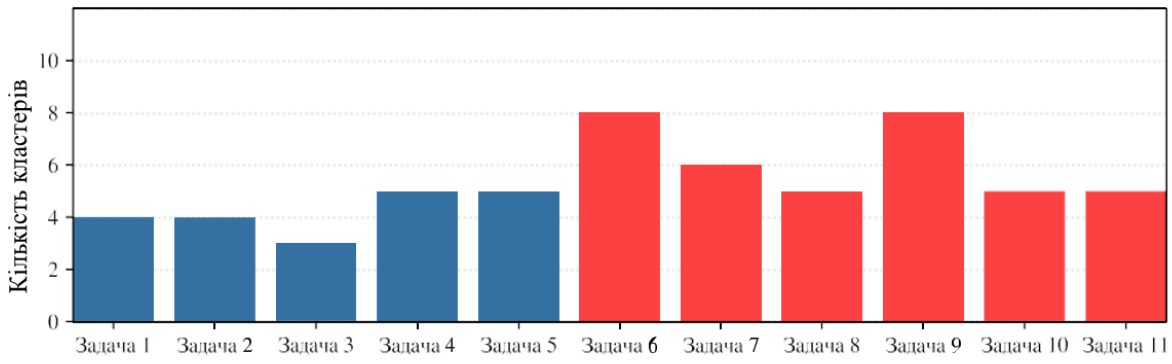


Рисунок 3.5 – Число кластерів після фільтрації для завдань 11 типів

Це дозволяє об'єднувати в один кластер рішення, що використовують різні форми записи циклів і різні структури даних для реалізації одного і того

ж способу розв'язання задачі. Без додаткових замін для завдань із 6 по 11 число малих кластерів значно зростає. Типи завдань у LMS ЦАВ ранжуються у порядку збільшення складності їх рішення. Як показано на рисунок 3.4 та в таблиці 3.6, з збільшенням складності завдань також зростає варіативність підходів до їх вирішення.

Таблиця 3.7 – Приклади програм, вирішальних завдання типу 3 різними способами

Завдання	Спосіб 1	Спосіб 2	Спосіб 3

3	def main(m, b, n):
	  s = 0
	  for k in range(1, n+1):
	  for i in range(1, b+1):
	  for j in range(1, m+1): s += i**6 + 65*k + j
	  return s
		def main(a, n):
		  x = sum(2*c for c in range(1, a+1))
		  y = sum(j ** 12 - 99 * j + j ** 3 for j in range(1, n+ 1))
		  return x + y
			def main(m, n, x):
			  j, i, s = 1, 1, 0
			  while j <= n:
			  while i <= m:
			  t = x/31 - 78*j**2 s += 40 * i ** 2 - 11 * ti += 1
			  i, j = 1, j + 1
			  return s

Як видно з таблиці 3.7, способи рішення завдань на реалізацію ітераційною формули включають рішення за допомогою вкладених циклів For, рішення за допомогою виразів-генераторів, рішення за допомогою циклу While. Способи вирішення задач на реалізацію рекурентної формули включають рішення за допомогою циклу і змінних, а також рішення при допомоги рекурсивний функції. Способи розв'язання задач на розбір двійкового формату даних включають рішення за допомогою класу BinaryReader зі прихованим станом, а також рішення за допомогою функцій, повертаючих прочитане значення і нове стан.

3.3 Визначення способів рішення завдань на основі класифікації програм

На п'ятому кроку методики розмічені по результатам кластеризації набори векторних подання текстів програм (таблиця 3.6) використовувалися для розробки класифікаторів, здатних визначати спосіб розв'язання задачі за векторним поданням тексту програми . Відомості о підготовлених для навчання класифікаторів набори пар, кожна з яких складається з векторного представлення тексту програми та відповідної йому мітки кластера, наданої за результатами кластеризації набору текстів програм, наведені в таблиці 3.8.

Таблиця 3.8 – Відомості о наборах даних, підготовлених для розробки класифікаторів

Тип завдання	1	2	3	4	5	6	7	8	9	10	11
Розмір розміченого набору	1312	1290	1211	1186	1233	1232	1245	1229	1231	1211	1174
Розмір навчального набору	918	903	847	830	863	862	871	860	861	847	821
Розмір тестового набору	394	387	364	356	370	370	374	369	370	364	353
Число класів	4	4	3	5	5	8	6	5	8	5	5

Розміри розмічених наборів, перелічені в таблиці 3.8, відрізняються від розмірів наборів програм, надісланих до ЦАВ (таблиця 3.5), оскільки програми, що належать кластерам, що містить мале число програм, були виключено при фільтрації на четвертому кроці методики автоматизованого обліку навчальних досягнень. Порогові значення мінімально допустимого числа програм, вхідних в кластер, представлені в таблиці 3.6. Число класів в таблиці 3.8 збігається з числом кластерів у таблиці 3.6 та на рисунок 3.4. При класифікації програми, в якій використовується новий спосіб розв'язання задачі, не відомий на етапі розробки класифікатора, завдання вважається вирішеною найбільш схожим способом з числа відомих класифікатору.

Було розроблено такі класифікатори, як KNN, SVM, RF, ELM. У процесі проведення експериментальних досліджень використовувався мова програмування Py-thon версії 3.10, а також методи і алгоритми багатокласовий класифікації KNN, SVM, RF, реалізовані в бібліотеці sklearn. ELM-класифікатор був реалізований за формулами та за допомогою матричних операцій, доступних у бібліотеці numpy. Вибір ELM-класифікатора обумовлений високою швидкістю його навчання по порівнянні зі швидкістю навчання в методі зворотного поширення помилки, що використовується в класичних ІНС. Вибір значень параметрів класифікаторів здійснювався на основі пошуку по сітці та перехресної перевірки якості класифікаторів в сенсі F_1-заходи по k блоків при $k = 5$. При виборі параметрів класифікаторів використовувалися всі програми з 11 розмічених наборів. Оптимальні значення параметрів KNN, SVM, RF, ELM, за яких було досягнуто максимальне значення F_1-заходи для кожного з 11 наборів векторних уявлень програм.

3.4.Автоматичний облік навчальних досягнень

Розроблені ELM-класифікатори, що дозволили досягти високою швидкості прогнозування і найкращої точності прогнозів після налаштування вхідних ваг і зрушень прихованого шару за допомогою алгоритму 2, були інтегровані в LMS ЦАВ [5] і використовувалися для обліку навчальних досягнень студентів курсу програмування мовою Python.

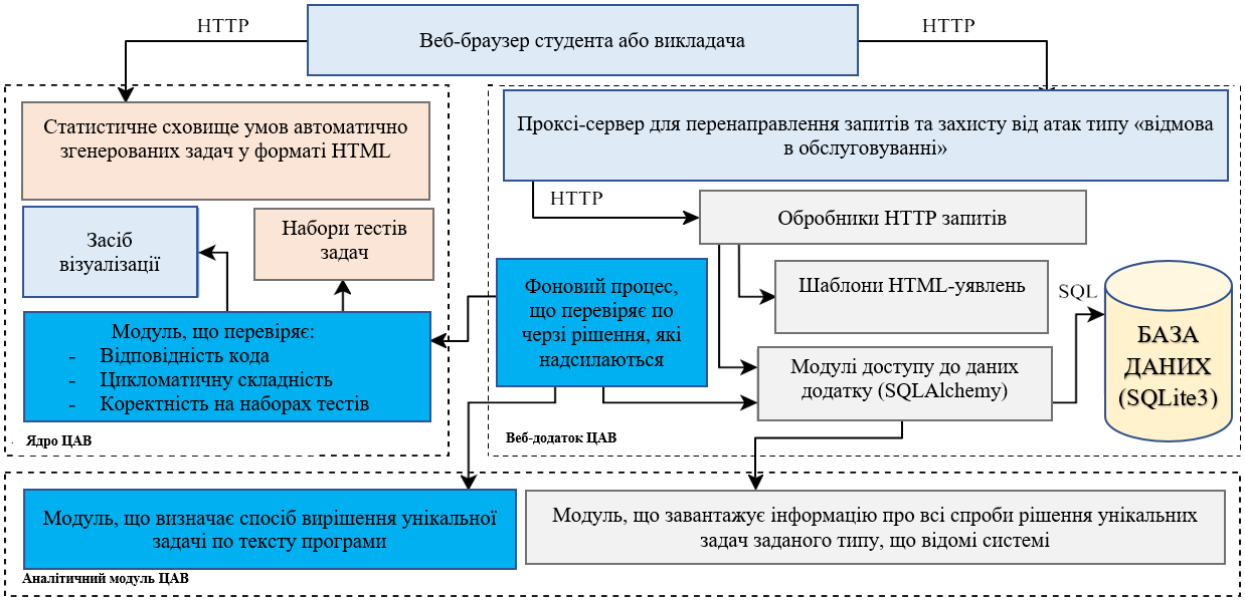


Рисунок 3.6 – Схема LMS ЦАВ з підтримкою автоматичного обліку навчальних досягнень

Оновлена архітектура LMS ЦАВ з підтримкою автоматичного обліку досягнень (рисунок 3.6) включає 3 модулі, таких як ядро, що підтримує автоматичне породження та перевірку унікальних завдань по програмування 11 типів, веб-додаток, надає інтерфейс користувача для перегляду списку груп, списку студентів, статусів відправлених рішень задач, а також аналітичний модуль, що виконує аналіз програм для визначення способів рішення задач. Аналітичний модуль зберігає відомості о способи вирішення задач, виявлені на етапі кластеризації на третьому кроці методики автоматизованого обліку навчальних досягнень, а також виконує статичний аналіз програм при використанні алгоритму 4 та розроблених ELM-класифікаторів векторних уявлень програм.

Студентам експерементальної групи, в якої використовується LMS ЦАВ, пропонується вирішити унікальні. Завдання всіма відомими аналітичному модулю способами рішення. Діаграма послідовностей для процесу взаємодії студента із системою LMS ЦАВ представлена рисунку 3.7.

Як показано на рисунку 3.7, після відправки рішення студентом ядро LMS ЦАВ перевіряє при сланий текст програми, потім програма аналізується

розробленим інструментом інтелектуального статичного аналізу для визначення способу рішення завдання.

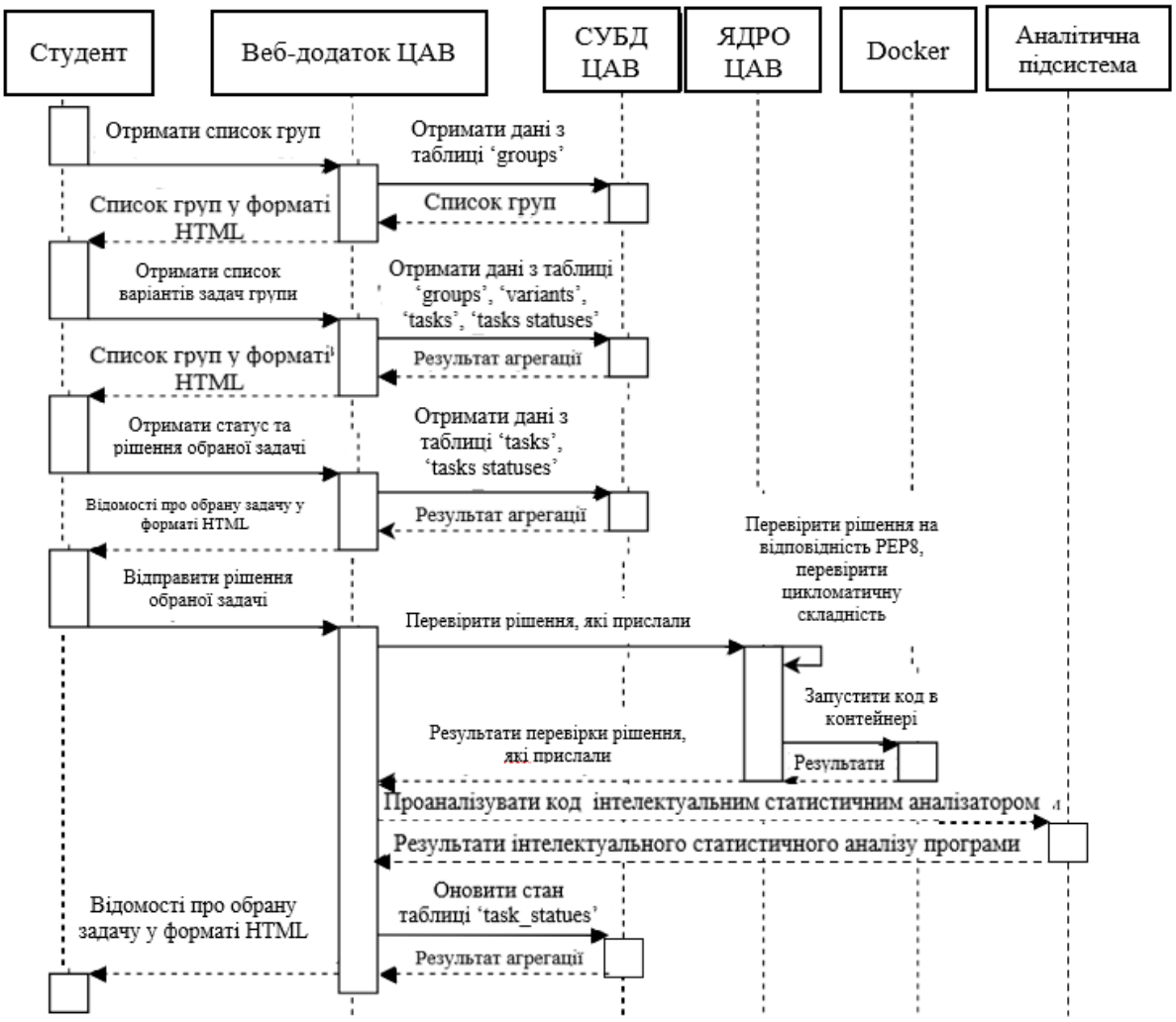


Рисунок 3.7 – Діаграма послідовностей процесу взаємодії студента з LMS ЦАВ

Після цього отримані відомості про спосіб розв'язання задачі зберігаються в базі даних LMS ЦАВ і відображаються в інтерфейсі користувача веб-додатку. Діаграма сутність-зв'язок для бази даних веб-додатки LMS ЦАВ показано на рисунку 3.8. База даних веб-додатки включає в себе 7 таблиць і забезпечує зберігання номерів завдань, відомостей о групах, номерах варіантів, відповідних порядковим номерам студентів, статусів завдань, містять інформацію про те, чи прийнято відправлене рішення. Крім того,

таблиця статусів задач також містить відомості про спосіб вирішення задач.

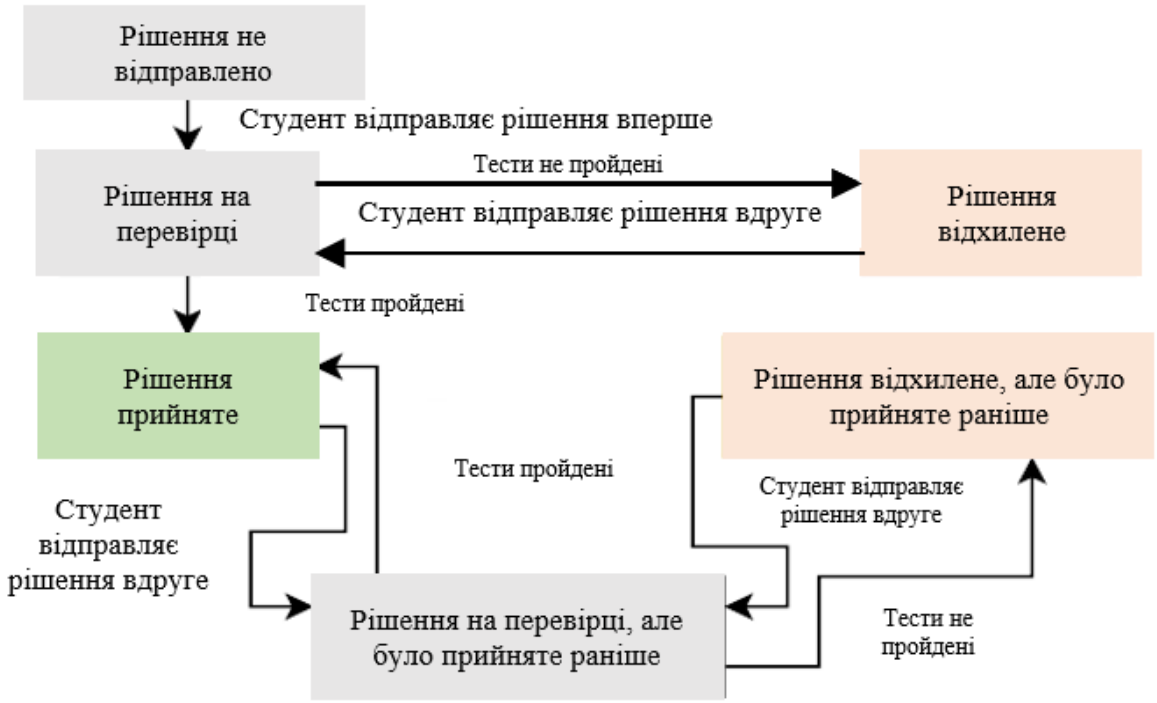


Рисунок 3.8 – Діаграма, що описує можливі стани завдань та переходи між ними

Як показано на рисунку 3.8, відомості про всіх досягнення, отриманих студентом при рішенні унікальної завдання, зберігаються в поле «Achievements» таблиці "task_statuses" бази даних веб-додатку LMS ЦАВ. Веб-інтерфейс для надсилання рішення задачі та відстеження результатів перевірки має вигляд, показаний на рисунок 3.7, у разі, якщо останнє відправлене рішення задачі прийнято системою і завдання було вирішено всіма способами, відомими аналітичному модулю LMS ЦАВ. У випадку, якщо програма була відхилена LMS ЦАВ, у веб-інтерфейсі наводяться подрібнені відомості о причини відхилення програми, такі як повідомлення про помилка, виникла в процесі виконання програми, або висновок статичного аналізатора. Діаграма, описує можливі стани завдань і переходи

між ними, представлена на рисунку 3.8.

Як показано на рисунку 3.8, для здачі унікальною завдання по програмування достатньо розв'язати задачу будь-яким із відомих системі способів. Наступні спроби розв'язання задач додатковим способами можуть привести або до відхилення рішення, або до прийняттю рішення новим способом з занесенням відомостей про відкритому способі рішення в поле «Achievements» таблиці "task_statuses" бази даних веб-додатки LMS ЦАВ (рисунок 3.7) з показом нових досягнень у інтерфейсі веб-додатка.

В результаті впровадження розробленої методики автоматизованого обліку навчальних досягнень у LMS ЦАВ для визначення способів розв'язання задач студенти груп, у яких бали за відкриті способи вирішення підсумовуються з балами за активність, вирішують у середньому на 6 задач більше додатковими способами. Результати кластеризації текстів програм використовуються викладачами для оцінки прогалин в навичках та знаннях учнів, а також для оцінки якості генераторів задач. У осінньому семестрі 2024-2025 навчального року в LMS ЦАВ було надіслано до 5000 програмних рішеннь автоматично згенерованих завдань, при цьому кількість віртуальних учнів, які вирішили все 11 обов'язкових навчальних завдань хоча б одним із способів, було одно 1100. У весняному семестрі 2024-2025 навчального року в LMS ЦАВ з підтримкою автоматичного обліку навчальних досягнень було надіслано 100 691 рішення, при цьому число віртуальних учнів, вирішили все 11 обов'язкових завдань хоча б одним із способів, дорівнювало 1053.

ВИСНОВОК

Розроблено і досліджено алгоритм перетворення текстів програм в векторні подання , засноване, на відміну від відомих аналогів, на побудові ланцюгів Маркова для AST. Виявлено, що векторні уявлення програм на основі ланцюгів Маркова для AST мають знижену чутливість до класифікатора в порівнянні з аналогами

Встановлено, що векторні уявлення програм на основі ланцюгів Маркова для AST дозволяють підвищити показник чистоти кластеризації і індекс Ренда в середньому на 40% у задачі розбиття набору програм на кластери залежно від реалізованих у програмах алгоритмів порівняно з аналогами, заснованими на word2vec та на обчисленні редакційних відстаней між AST програм, в випадку спільного використання векторних уявлень програм на основі ланцюгів Маркова для AST та алгоритму ієрархічної кластеризації за методом середнього зв'язку, при умові обчислення попарних відстаней між програмами на основі відстані Єнсена-Шеннона.

Розроблено метод винесення рекомендацій по підвищення швидкості виконання програм, що зводиться до формування бази програмних реалізацій алгоритмів, вимірів швидкості виконання оригінальних реалізацій алгоритмів і реалізацій, прискорених за допомогою доступних на заданій гетерогенній обчислювальній платформі спеціалізованих прискорювачів, пошуку фрагментів AST за програмою-прикладом, створенням звіту по перспективам прискорення ПС з використанням отриманих чисельних оцінок.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Кучук Г. А., Ірха Д. А., Кириленко Д. В. Аналіз розташування об'єкта доповненої реальності у фізичному просторі. Системи управління, навігації та зв'язку. Полтава : Національний університет «Полтавська політехніка імені Юрія Кондратюка», 2025. Вип. 2(80). С. 150–156.

2. Hua, W.FCCA: Hybrid code representation for functional clone detection using attention net-worksW. H ua, Y. Sui, Y. Wan, G. Liu, G. Xu // IEEE Transactions on Reliability. – 2021. – V. 70, №1. – P. 304 – 318.

3. Jiang, L.Deckard: Scalable and accurate tree-based detection of code clones / L. Jiang, G. Misherghi, Zh. Su, S. Glondu //Proceedings of the 29th International Conference on Software Engineer-ing (ICSE'17). – 2017. – P. 96 – 105.

4. Zhang, J. A novel neural source code representation based on abstract syntax tree / J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, X. Liu //In Proceedings of the 41st International Conference on Software Engineering (ICSE'19). –2019. – P. 783–794.

5. Jo,Y.Two-Pass technique for clone detection and type classification using tree-based convolu-tion neural network / Y. Jo, J. Lee, C. Yoo //Applied Sciences. – 2021.– V.11,№14.–P. 6613.

6. Sripada, S.C. Comparison of purity and entropy of k-means clustering and fuzzy c means clus-tering / S.C. Sripada, M.S. Rao // Indian journal of computer science and engineering. – 2011. – V. 2, № 3. – P. 343 – 346.

7. Rand, W.M. Objective criteria for the evaluation of clustering methods / W.M. Rand // Journal of the American Statistical association. – 1971. – V. 66, № 336. – P. 846 – 850.

8. Sokal, R.R.A Statistical Method for Evaluating Systematic Relationships/ R.R. Sokal, C.D. Michener //Evolution. – 1957. – V.11. – P. 130 – 162.

9. Griffiths, A. Hierarchic agglomerative clustering methods for automatic

document classification / A. Griffiths, L.A. Robinson, P. Willett //Journal of Documentation. – 1984. – V 40, №3. – P. 175 – 205.

10. Zhang, K. Simple fast algorithms for the editing distance between trees and related problems / K. Zhang, D. Shasha // SIAM Journal on Computing. – 1989. – V. 18, № 6. – H. 1245 – 1262.

11. Mirjalili, S. Grey wolf optimizer[Текст] / S. Mirjalili, S. M. Mirjalili, A. Lewis//Advances in engineering software. – 2014. – V.69. – P. 46 – 61.

12. Bastos Filho,C.J.A. A novel search algorithm based on fish school behavior / C.J.A. Bastos Filho, F.B. de Lima Neto, A.J.C.C. Lins, A.I.S. Nascimento, M.P. Lima //2008 IEEE international conference on systems, man and cybernetics. – IEEE, 2008. – P. 2646 – 2651.

13. Kudela,J. The Evolutionary Computation Methods No One Should Use[Текст]/J. Kudela //Xiv preprint. – Xiv, 2023. – P. 301.

14. Lai,G.A LPSO-SGD algorithm for the Optimization of Convolutional Neural Network /G. Lai, F.Li,J.Feng, Sh. Cheng// IEEE Congress on Evolutionary Computation (CEC). – 2019. – P. 1038 – 1043.