Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

Галузь знань 12 – Інформаційні технології

Спеціальність 123 –Комп'ютерна інженерія

на тему «Метод створення SVM-класифікатора для аналізу даних на базі FPGA»

КвРКІП. 182122.22.01.41 ПЗ

Виконала: студентка 2 курсу, група КІ2м-22-1 _____ Шпуляр Є. М.
Підпис Ініціали, прізвище

Керівник к.т.н. доцент _____ Іванов О. В.
Науковий ступінь, вчене звання Підпис Ініціали, прізвище

До захисту допускаю:
Зав. кафедри КІІС, д.т.н., проф.
Т.О. Говорущенко
14 05 2024 р.

Хмельницький, 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Освітній рівень МАГІСТР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма ОСВІТНЬО-НАУКОВА ПРОГРАМА «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

" 01 " 09 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Шпуляр Євгенії Миколаївні

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Метод створення SVM-класифікатора для аналізу даних на базі FPGA

Керівник проекту (роботи) Іванов О. В. к.т.н. доцент

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.01.2024 р. № 1

2. Строк подання студентом проекту (роботи) на кафедру 01.05.2024 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Аналіз відомих методів класифікаторів машинного навчання та їх застосування

Розробка оптимального методу створення SVM-класифікатора для FPGA

Реалізація та тестування розробленого методу на FPGA

Оцінка ефективності та порівняння з існуючими підходами

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

6. Консультанти розділів кваліфікаційної роботи магістра

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Лисенко С.М., професор кафедри КІІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІІС		

7. Дата видачі завдання «_01_»_____09_____2023р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) кваліфікаційної роботи магістра	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики КвРМ з керівником	01.09.2023	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.10.2023	виконано
3	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	01.11.2023	виконано
4	Робота над розділом 2 – розробка моделей для вирішення поставленої задачі	01.12.2023	виконано
5	Робота над науковою статтею	01.02.204	виконано
6	Робота над розділом 3 – розробка методів для вирішення поставленої задачі	15.02.2024	виконано
7	Робота над розділом 4 – проектування та розробка ПЗ для вирішення поставленої задачі, експериментальна частина	01.04.204	виконано
8	Оформлення пояснювальної записки згідно вимог	18.04.2024	виконано
9	Попередній захист ДРМ	29.04.2024	виконано
10	Захист ДРМ на засіданні ЕК	До 15.05.2024	

Студентка _____ Шпуляр Є. М.
Підпис Ініціали, прізвище

Керівник роботи _____ Іванов О. В.
Підпис Ініціали, прізвище

РЕФЕРАТ

Тема кваліфікаційної роботи магістра: Метод створення SVM-класифікатора для аналізу даних на базі FPGA.

Автор роботи: Шпуляр Є. М.

Керівник роботи: Іванов О.В.

Пояснювальна записка: 85 с., 23 рис., 7 табл., 2 дод., 79 джерел.

Ключові слова: SVM-класифікатор, машинне навчання, опукла оптимізація, FPGA, методи аналізу даних, класифікація даних, метод опорних векторів..

Об’єктом дослідження є SVM-класифікатор для аналізу даних на базі FPGA.

Предметом дослідження є метод створення SVM-класифікатора для аналізу даних на базі FPGA.

Метою кваліфікаційної роботи магістра є підвищення ефективності виконання аналізу даних шляхом інтеграції SVM-класифікатора з FPGA для реалізації системи

Для розв’язання поставлених задач використовуються основні положення теорії штучного інтелекту, системного аналізу, моделювання, методів аналізу даних, теорії математичної статистики, теорії дискретної математики.

Наукова новизна отриманих результатів:

− запропоновано удосконалений метод створення SVM-класифікатора для аналізу даних, який на відміну від відомих здійснює аналіз даних за допомогою застосування апаратних можливостей FPGA, що уможливило забезпечити зменшення часу аналізу даних, підвищення точності, зменшення програмно-апаратних ресурсів;

− розроблено програмно технічний засіб інтеграції SVM-класифікатора на базі FPGA для аналізу даних, який полягає в тому, що використовується апаратна реалізація математичних операцій для швидкого обчислення.

На основі проведених досліджень розроблено вбудований апаратний дизайн з урахуванням специфіки завдань аналізу даних, що дозволило створити

ефективний інструмент для реалізації SVM-класифікатора на основі опуклої оптимізації на FPGA

Практична значимість отриманих результатів полягає у тому, що в результаті виконаного наукового дослідження розроблений метод створення SVM-класифікатора для аналізу даних на базі FPGA може бути застосований у різних областях, де потрібно виконувати класифікацію в реальному часі та обробляти великі обсяги даних.

У першому розділі досліджено можливості використання FPGA для аналізу даних. Було проаналізовано різні класифікатори машинного навчання, їх переваги та недоліки. Зокрема, в результаті проведеного аналізу виявлено, що існуючі підходи до реалізації класифікаторів на базі FPGA мають такі недоліки, як обмежена швидкодія, обмежені можливості обробки складних алгоритмів та обмежені ресурси пристрою. Таким чином, при повній відсутності подібної роботи в опублікованій літературі, яка б надала апаратні архітектури на основі FPGA для SVM методу основаного на опуклій оптимізації на вбудованих пристроях, було виявлено потребу в розвитку оптимізованого та удосконаленого методу створення класифікатора на базі FPGA, який би ефективно використовував можливості цього пристрою та забезпечував високу точність результатів.

У другому розділі було досліджено та проаналізовано важливі аспекти оптимальної гіперплощини, яка є центральним елементом у методі опорних векторів. Було досліджено методи нелінійної оптимізації, які використовуються для побудови нелінійних розділяючих гіперплощин у випадках, коли дані не є лінійно роздільними. Було досліджено опуклу оптимізацію як ключовий компонент методу опорних векторів. Виявлено, що використання опуклої оптимізації дозволяє ефективно знаходити глобальний мінімум функції втрат для оптимального розташування гіперплощини у просторі.

У третьому розділі було досліджено та проаналізовано основи створення SVM-класифікатора для аналізу даних на базі FPGA. Виявлено, що використання методу опорних векторів у поєднанні з опуклою оптимізацією дозволяє ефективно розв'язувати задачі класифікації навіть для складних та нелінійних наборів даних.

Архітектура на рівні системи була детально проаналізована з метою розробки оптимізованої системи аналізу даних на базі FPGA. Для реалізації запропонованого методу було створено вбудований апаратний дизайн з урахуванням специфіки завдань аналізу даних, що дозволило створити ефективний інструмент для реалізації SVM-класифікатора на основі опуклої оптимізації на FPGA.

У четвертому розділі в результаті проведеного експериментального дослідження було виявлено, що запропонований вбудований апаратний прискорювач для SVM методу на основі опуклої оптимізації виконувався до 79 разів швидше, ніж його програмний еквівалент, який працював на вбудованих мікропроцесорах. Запропоновані вбудовані рішення (як апаратні, так і програмні) досягли до 100% точності класифікації та кращої продуктивності (тобто, за постійної кількості ітерацій для пошуку мінімуму під час оптимізації, потребували набагато менше ітерацій) порівняно з кодом Python на настільних комп'ютерах.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

SVM - Support Vector Machine (метод опорних векторів)

FPGA - Field Programmable Gate Array (вентильна програмована матриця)

SDRAM - Synchronous dynamic RAM (синхронна динамічна пам’ять з довільним доступом)

CLB - Configurable logic blocks (програмовані логічні блоки)

HDL - Hardware description languages (мова опису апаратури)

MCU - Micro Controller Unit (мікроконтролер)

ASIC - Application-specific integrated circuit (інтегральна схема для специфічного застосування)

GPU - Graphics processing unit (графічний процесор)

CPU - Central processing unit (центральний процесор)

SGD - Stochastic gradient descent (стохастичний градієнтний спуск)

HFK - Hardware-friendly kernel (ядро, оптимізоване для апаратного забезпечення)

RPNs - Reconfigurable processing nodes (переконфігуровувані вузли обробки)

AXI - Advanced Extensible Interface (розширений розширюваний інтерфейс)

CO - Convex Optimization (опукла оптимізація)

IP - Intellectual Property (інтелектуальна власність)

IPIF - AXI Intellectual Property Interface (інтелектуальна Власність Інтерфейсу AXI)

IPIC - Intellectual Property Interconnects (інтелектуальна власність міжз’єднань)

FSM - Finite State Machines (скінченні автомати)

RAM - Random Access Memory (оперативна пам’ять, пам'ять з довільним доступом)

ROM - Read-Only Memory (пам'ять постійного зберігання)

BRAM - Block Random Access Memory (блоки пам’яті)

RBF – Radial basis function (радіальна базисна функція)

MAC – Multiply-accumulate (пристрій множення та додавання)

SMO - Sequential Minimal Optimization (послідовна мінімальна оптимізація)

ANN - Artificial Neural Network (штучні нейронні мережі)

DM-SNR - Dispersion Measure- Signal to weight Ratio (міра дисперсії - співвідношення сигнал/вага)

ПЗ - програмне забезпечення

ВСТУП

У сучасному світі зростає обсяг та складність даних, що потребує ефективних методів їх аналізу та обробки. Машинне навчання, зокрема методи класифікації, стає ключовим інструментом для автоматизованої обробки та аналізу даних у різних сферах, від медицини до фінансів. Однак, зростаюча потреба у швидкодії та енергоефективності вимагає нових підходів до реалізації цих методів. Одним з ключових інструментів для класифікації та аналізу даних є метод опорних векторів (SVM - Support Vector Machine), який широко використовується у бізнесі, науці, медицині та багатьох інших галузях.

Використання методу створення SVM-класифікатора для аналізу даних на базі FPGA є актуальною проблемою, оскільки FPGA забезпечують високу швидкодію та енергоефективність, що робить їх привабливими для застосування в області машинного навчання.

Незважаючи на свою ефективність, реалізація SVM-класифікатора може стикатися з викликами, пов'язаними з обмеженими ресурсами та швидкодією обробки даних. У цьому контексті виникає необхідність розвитку нових методів інтеграції SVM-класифікаторів з високопродуктивним обчислювальним обладнанням.

Актуальність роботи полягає в розробці методу створення SVM-класифікатора для аналізу даних на базі FPGA, що відомі своєю високою швидкодією та можливістю паралельної обробки даних, що робить їх ідеальним кандидатом для ефективної реалізації класифікаторів машинного навчання.

У цій роботі розглядаються основні принципи роботи класифікаторів машинного навчання, зокрема SVM-класифікатор, описані відомі методи реалізації на FPGA, а також їх переваги та недоліки.

Проте, розробка ефективного та оптимізованого методу створення SVM-класифікатора для FPGA вимагає додаткових досліджень та розробки, оскільки існуючі методи можуть бути неоптимальними з точки зору швидкодії та використання ресурсів FPGA.

Дослідження даної теми є важливим з огляду на широкий спектр застосувань методів класифікації в різних областях, а також на потребу у швидкодії та енергоефективності, що є критичними в сучасних обчислювальних системах. Ефективний SVM-класифікатор на базі FPGA може знайти застосування в медичній діагностиці, фінансовому аналізі, обробці сигналів та багатьох інших областях, де вимагається аналіз великих обсягів даних у реальному часі.

Метою даної роботи є підвищення ефективності виконання аналізу даних шляхом інтеграції SVM-класифікатора з FPGA для реалізації системи.

Поставлена мета досягається розв’язанням таких основних задач:

- розглянути застосування та реалізації FPGA;
- дослідити відомі класифікатори машинного навчання;
- проаналізувати існуючі методи реалізації класифікаторів;
- необхідно розробити оптимальний метод створення SVM-класифікатора для FPGA;
- реалізувати та протестувати розроблений метод на FPGA;
- оцінити ефективність та порівняти з існуючими підходами;

Об’єктом дослідження є SVM-класифікатор для аналізу даних на базі FPGA.

Предметом дослідження є метод створення SVM-класифікатора для аналізу даних на базі FPGA.

Наукова новизна отриманих результатів:

1. Запропоновано удосконалений метод створення SVM-класифікатора для аналізу даних, який на відміну від відомих здійснює аналіз даних за допомогою застосування апаратних можливостей FPGA, що уможливило забезпечити зменшення часу аналізу даних, підвищення точності, зменшення програмно-апаратних ресурсів.

2. Розроблено програмно технічний засіб інтеграції SVM-класифікатора на базі FPGA для аналізу даних, який полягає в тому, що використовується апаратна реалізація математичних операцій для швидкого обчислення.

Практична цінність отриманих результатів. В результаті виконаного наукового дослідження розроблений метод створення SVM-класифікатора для

аналізу даних на базі FPGA може бути застосований у різних областях, де потрібно виконувати класифікацію в реальному часі та обробляти великі обсяги даних. Отриманий метод є удосконаленим та оптимізованим порівняно з іншими підходами, забезпечуючи значне покращення швидкості та ефективності аналізу даних, що робить його більш привабливим для використання в різних застосуваннях. Ця оптимізація дозволяє досягти великої точності класифікації при значно менших часових затратах на обробку даних. Таким чином, розроблений метод має великий потенціал для успішного впровадження в реальних умовах, в таких сферах, як:

1. Медична діагностика: швидка та точна класифікація медичних зображень для виявлення захворювань, таких як рак або неврологічні захворювання.

2. Контроль якості: виявлення дефектів на виробничих лініях або виробництво з великою швидкістю та точністю.

3. Обробка великих обсягів даних: аналіз великих даних для виявлення закономірностей, трендів та патернів у різних галузях, включаючи фінанси, телекомунікації, транспорт тощо.

4. Розпізнавання образів: класифікація об'єктів на зображеннях або відеозаписах для автоматичного виявлення об'єктів, розпізнавання лиць, відстеження об'єктів тощо.

Для розв’язання поставлених задач використовуються основні положення теорії штучного інтелекту, системного аналізу, моделювання, методів аналізу даних, теорії математичної статистики, теорії дискретної математики.

За темою кваліфікаційної роботи магістра опублікована одна стаття у фаховому науковому журналі «Computer systems and information technologies» у 2024 році [1].

1 ДОСЛІДЖЕННЯ ВІДОМИХ МЕТОДІВ АНАЛІЗУ ДАНИ НА БАЗІ FPGA

1.1 Застосування FPGA

FPGA (Field-Programmable Gate Array) є логічною інтегральною схемою, що програмується користувачем. Це програмований в полі масив логічних вентилів, що дозволяє перепрограмувати апаратне забезпечення після його виробництва. Основна перевага FPGA полягає в їх гнучкості: вони можуть бути швидко перепрограмовані для виконання різних завдань, від обчислення до обробки сигналів до алгоритмів шифрування. FPGA часто використовуються в сферах, де потрібна висока швидкість обчислень та низька затримка, таких як вбудовані системи, мережеві пристрої, обробка сигналів та багато іншого. Іншими словами, вони надають можливість програмування апаратного забезпечення на рівні логічних елементів, що робить їх вельми корисними для широкого спектру застосувань [2,3].

FPGA не використовує чітко задану архітектуру, як будь-який інший процесор, тому доводиться будувати її самостійно на початковому етапі реалізації проєкту. Це дає можливість конфігурації потрібного функціоналу і збірки його в необхідну архітектуру, щоб мати змогу вирішувати конкретну задачу. Більш того, використання FPGA дозволяє здійснення реалізації функцій паралельно, що означає можливість виконання кожної задачі в незалежності від інших. Єдине обмеження – побудова алгоритму задачі, під час якого для початку одного процесу необхідно отримати вихідні дані іншого. Ця унікальність вирізняє FPGA перед іншими типами процесорів.

У загальному, при побудові FPGA виділяються три головні елементи: програмовані логічні блоки (CLB, Configurable logic blocks), інтерфейсні блоки вводу-виводу (Input/Output blocks) та матриці внутрішніх зв’язків (Interconnections matrix).

Логічні елементи – основна "сердечна" частина FPGA, яка складається з комбінаційних та регістрів пам'яті. Комбінаційні логічні елементи виконують

логічні операції, такі як AND, OR, NOT, а регістри пам'яті використовуються для зберігання даних [6]. Логічними блоками є базові функціональні елементи, які можуть бути сконфігуровані в потрібні логічні функції, елементи пам’яті чи спеціалізоване ядро, що дає можливість виконати чітко визначену задачу. Завдяки технологічному рівні FPGA і визначається набори можливих конфігурацій блоку.

FPGA містять вбудовані блоки пам'яті, такі як блоки RAM (Random Access Memory) та блоки ROM (Read-Only Memory). Ці блоки можуть використовуватися для зберігання даних або програм [76].

Мікросхеми – це набір провідників та мікросхем, які з'єднують логічні елементи та блоки пам'яті FPGA. Вони визначають фізичні шляхи для передачі сигналів між різними частинами FPGA.

Конфігураційна пам'ять – це пам'ять, в якій зберігається конфігураційна інформація FPGA, така як програма або дизайн, який потрібно виконати. Ця інформація завантажується при запуску FPGA.

FPGA мають піни вводу-виводу, які дозволяють взаємодіяти з зовнішнім світом. Ці піни можуть бути налаштовані для введення або виведення цифрових сигналів, а також для роботи з різноманітними інтерфейсами, такими як HDMI, USB, Ethernet та інші [3]. Блоки вводу-виводу забезпечують зв’язок виводів корпусу та внутрішніх сигнальних ліній. Ці блоки налаштовуються в залежності від типів інтерфейсів, що необхідні при комунікації різних периферійних елементів.

Матриця внутрішніх зв’язків слугує для об’єднання різних функціональних блоків в довільних конфігураціях.

Ці складові частини працюють разом, щоб програмно налаштувати FPGA для виконання певних завдань, забезпечуючи гнучкість та швидкість, які часто роблять FPGA вигідними у порівнянні з традиційними спеціалізованими мікросхемами.

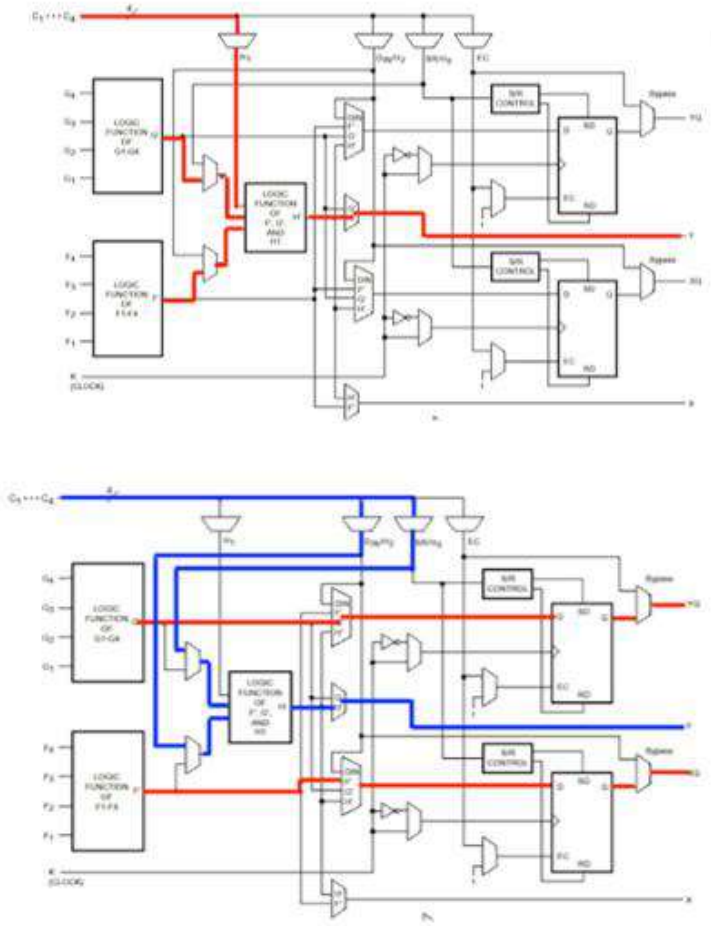


Рисунок 1.1 – Варіанти конфігурацій блоку

Дві різних конфігурації того ж самого блоку представлено на рисунку 1.1. Результат отримання різних функціональних блоків визначається вибором налаштувань мультиплексорів. Файл програми є набором конфігурацій ключів програмованих блоків.

Представлене на рисунку 1.2 віртуальне зображення реального проєкту дає можливість побачити поширення тих чи інших сигналів та їх використання всередині чіпа.

FPGA програмується використовуючи спеціалізовані комп'ютерні мови HDL (hardware description languages), які є мовами опису апаратури. Розробники можуть не використовувати опис конкретних елементів схеми, а займатись програмуванням поведінки системи в цілому або конкретних блоків [5,6]. Найбільш популярними мовами є VHDL та Verilog.

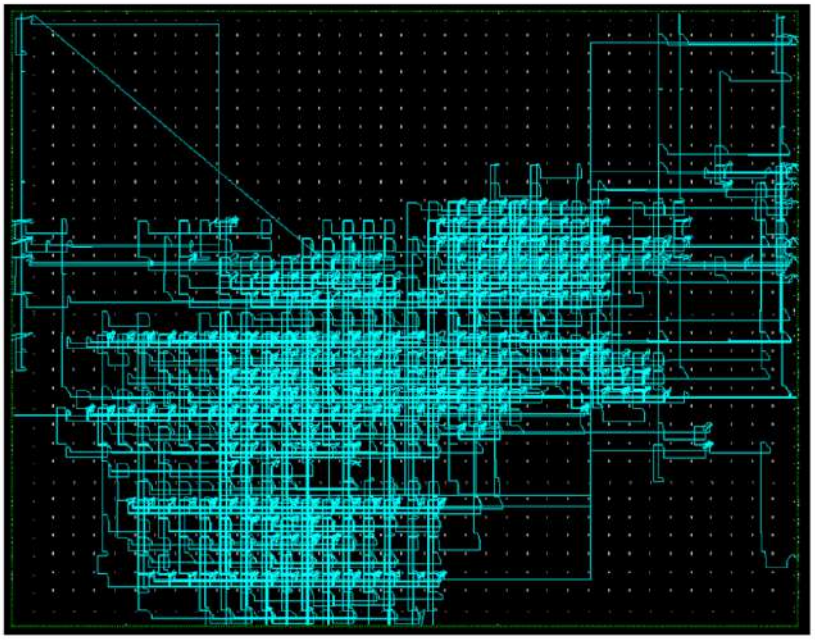


Рисунок 1.2 – Віртуальне зображення реального проєкту.

Під час програмування розробники мають можливість моніторингу поведінки системи та функціоналу в момент часу при тих чи інших даних, що були отримані на вході. Це дає змогу до глибокого проникнення у реалізований функціонал, докладних знань про зміни, що відбуваються, – все це має велику користь під час процесу створення програм.

Основними типами процесорів, що використовуються у вбудованих системах є [66,67]:

MCU — являються мікроконтролерами, що мають наперед закладену архітектуру. Це дає можливість гнучкого забезпечення рішень для певних задач.

Спеціалізовані інтегральні схеми (ASIC) — є мікросхемами спроєктованими спеціально для того, щоб виконати той чи інший функціонал. Їх серійне виробництво є значно дешевшим на відміну від інших типів контролерів. Але, зазвичай, вони мають потребу у доволі значних вкладеннях при використанні в велико серійних приладах.

Так, як різні типи мікросхем мають можливість виконання певного набору конкретних функцій [7], важливим є розуміння випадків, коли саме користування FPGA має найбільшу ефективність.

Як вже було сказано, унікальністю FPGA є можливість реалізації функцій паралельно [8]. Однак, не кожна задача може бути розбита на паралельні процеси. До того ж, можливість використати паралельні алгоритми не завжди оптимальне рішення.

При обробці відео та великих об’ємів інформації у реальному часі використання FPGA є незамінним. Використовуючи FPGA у алгоритмах мультимедійних аплікацій, можна забезпечити значно більшої продуктивності, на відміну від інших типів процесорів.

Прилади контролю за великими промисловими пристроями – є випадком, де є необхідність швидкого реагування на реального часу події, що стаються навколо. Використання FPGA дозволяє контролювати кожний момент часу всередині процесора (контролера), а це має надзвичайну користь.

FPGA також мають широкий спектр застосувань у різних галузях технологій і промисловості. FPGA часто використовуються для розробки вбудованих систем, таких як медичні прилади, автомобільна електроніка, мережеві пристрої тощо. Їх гнучкість дозволяє швидко адаптувати функціональність вбудованих систем до змінних вимог [69,70].

FPGA можуть виконувати обчислення в реальному часі для обробки аудіо-, відео- та інших сигналів. Вони широко використовуються у телекомунікаціях, радіо, а також у медичних системах, де потрібна обробка сигналів.

FPGA можуть бути використані для реалізації алгоритмів машинного навчання та глибокого навчання. Їх висока швидкодія та паралельна обробка роблять їх привабливими для виконання складних обчислень у реальному часі.

FPGA можуть використовуватися для реалізації різноманітних криптографічних алгоритмів, таких як шифрування даних, підписи, аутентифікація тощо. Їх гнучкість дозволяє легко змінювати криптографічні протоколи або параметри без необхідності зміни апаратного забезпечення.

FPGA можуть використовуватися для розробки та валідації апаратного забезпечення перед виробництвом. Їх можна програмувати для емуляції різних

апаратних конфігурацій або для перевірки правильності роботи апаратного забезпечення [6].

Кожний конкретний випадок, де застосовується FPGA, зумовлюється специфікою задачі.

Початок роботи з FPGA залежить від вибору контролера, що має оптимальний набір функцій, які вирішують конкретну задачу. Лідери виробників FPGA – це Xilinx, Intel(Altera) та Lattice. Коливання цін від 15-20$ до тисяч доларів мають залежність із задачами, які потребують реалізації з використанням FPGA. Обравши вендора, наступний крок – це вибір Development kit, навчальної плати, що буде використана для тестування створеного проєкту.

1.2 Класифікатори машинного навчання

Класифікатори машинного навчання - це алгоритми, які використовуються для автоматичного призначення категорій (класів) до об'єктів на основі вхідних даних. Основна мета класифікаторів полягає в тому, щоб вивчити залежності між вхідними ознаками (факторами) та вихідними мітками (класами) на основі навчальних даних, і використовувати ці знання для класифікації нових об'єктів, які раніше не були бачені моделлю [9, 17].

Отже, класифікатор машинного навчання може бути розглянутий як функція, яка відображає вхідні дані на вихідні класи. Ця функція може бути представлена у вигляді математичної моделі, яка може бути навчена на основі навчальних даних.

Для того щоб класифікатори працювали ефективно, вони навчаються на вхідних даних, які містять об'єкти з відомими мітками класів. Після навчання модель може бути використана для передбачення класів нових об'єктів, які раніше не бачилися. Процес навчання може включати оптимізацію параметрів моделі таким чином, щоб досягти найкращої точності класифікації на тестових даних [60].

Класифікатори машинного навчання можуть бути використані в різних областях, таких як розпізнавання образів, обробка природної мови, медична

діагностика, фінансовий аналіз та багато інших. Вони є важливим інструментом у сфері аналізу даних та розвитку інтелектуальних систем [9,10].

Лінійні класифікатори - це алгоритми машинного навчання, які використовують лінійну функцію для розділення простору ознак на класи.

Лінійна регресія – використовується для прогнозування числових значень на основі лінійної залежності між вхідними ознаками та цільовими значеннями.

Логістична регресія – використовується для бінарної класифікації, де модель прогнозує ймовірність того, що об'єкт належить до певного класу, і використовує логістичну функцію для перетворення вагованих сум в ймовірність.

Метод опорних векторів (Support Vector Machine, SVM) – шукає оптимальну розділюючу гіперплощину у просторі ознак, яка найкраще розділяє дані на класи [12, 15].

Усі ці класифікатори мають спільну властивість - вони використовують лінійну функцію для розділення простору ознак. Це означає, що вони ефективно працюють у випадках, коли вибірки даних можуть бути розділені лінійно. Однак, вони можуть мати обмежену ефективність у випадках, коли дані не розділяються лінійно.

Лінійні класифікатори є дуже популярними через свою простоту та ефективність у багатьох задачах класифікації та регресії. Їх можна легко застосовувати до різних типів даних та завдань, і вони є хорошим вибором для початкового аналізу даних [63].

SVM (Support Vector Machine) - це потужний і ефективний алгоритм, який зазвичай працює добре навіть для наборів даних з великою кількістю ознак. Однак правильний вибір параметрів і типу ядра є важливими для досягнення оптимальних результатів [16].

Головна мета SVM - знайти оптимальну розділювальну гіперплощину, яка максимізує відстань (зазвичай називається "зазором") між двома класами даних. Точки даних, які лежать на межі зазору та визначають положення розділювальної гіперплощини, називаються опорними векторами. Ці вектори є ключовими для

побудови класифікатора SVM. Після побудови моделі, нові дані класифікуються на основі того, на якому боці розділювальної гіперплощини вони знаходяться.

SVM може ефективно використовуватися для розділення нелінійних класів, завдяки концепції ядерних функцій. Ці функції дозволяють перетворювати нелінійні простори даних у простори більшої вимірності, де вони можуть бути лінійно розділені.

Наприклад, поліноміальне ядро використовується для трансформації даних у простори вищої вимірності за допомогою поліномів, в той час як радіальне базисне ядро (RBF) використовує гаусівську функцію для перетворення даних.

Модель SVM оптимізується шляхом максимізації зазору між двома класами та мінімізації функції втрат (наприклад, штраф за неправильну класифікацію). Цей процес може бути виражений у вигляді задачі квадратичного програмування (QP), яка може бути розв'язана за допомогою методів оптимізації [64].

SVM має деякі параметри, які можуть впливати на результат класифікації, такі як тип ядра (наприклад, лінійне, поліноміальне, радіальне), параметр регуляризації (C), який контролює компроміс між м'якістю та точністю, та параметри ядра (наприклад, степінь полінома чи ширина гаусівської функції).

Переваги SVM включають високу точність класифікації, ефективність в просторах великої вимірності та здатність працювати зі складними нелінійними відображеннями за допомогою ядр. Однак, він може бути чутливим до параметрів, таких як вибір ядра та налаштування параметрів регуляризації, і може бути обчислювально-витратним для великих обсягів даних.

Ще одним класифікатором машинного навчання є K-найближчих сусідів (k-NN). Це простий та ефективний алгоритм машинного навчання, який використовується для класифікації та регресії. Основна ідея полягає в тому, щоб класифікувати об'єкт на основі класів його k найближчих сусідів у просторі ознак.

K-найближчих сусідів використовується з параметром k, який визначає кількість найближчих сусідів, які беруться до уваги при класифікації нового об'єкта. Для визначення найближчих сусідів використовується відстань між об'єктами у просторі ознак. Зазвичай використовується Евклідова відстань, але

можуть використовуватися інші метрики відстаней. Після визначення k найближчих сусідів, об'єкт класифікується як більшість класів цих сусідів. Наприклад, якщо серед k сусідів більше належить до класу "A", то новий об'єкт буде класифікуватися як "A".

Вибір правильного значення k може впливати на результат класифікації. Малі значення k можуть призвести до перенавчання, тоді як великі значення k можуть призвести до недооцінки.

Вибір відповідної функції відстані також може впливати на результат класифікації. Хоча Евклідова відстань найчастіше використовується, інші метрики, такі як мангаттанська відстань або косинусна схожість, також можуть бути ефективними, особливо в залежності від характеру даних.

Перевагами класифікатору машинного навчання k-NN є простота та легкість реалізації. k-NN є непараметричним методом, що означає, що він не робить певних припущень про розподіл даних. k-NN враховує структуру даних, оскільки він використовує найближчі сусіди для класифікації, що дозволяє краще зберегти контекст даних.

Машинний класифікатор k-NN - це простий, але потужний метод класифікації, який може бути ефективним у багатьох сценаріях застосування. Однак важливо правильно вибирати параметр k та функцію відстані для досягнення оптимальних результатів.

Штучні нейронні мережі (ANN), або просто нейронні мережі, є одним з найбільш потужних та універсальних класифікаторів машинного навчання.

ANN складаються з набору шарів, включаючи вхідний, схований та вихідний. Сховані шари містять нейрони, які виконують обчислення на основі вхідних даних та ваг, що відповідають зв'язкам між нейронами у різних шарах.

Кожен нейрон у мережі використовує функцію активації для перетворення зваженої суми вхідних сигналів у вихідний сигнал. Різні функції активації можуть використовуватися в залежності від завдання та архітектури мережі. ANN навчаються шляхом адаптації ваг між нейронами у мережі з метою зменшення помилки між очікуваними та фактичними вихідними сигналами. Цей процес може

включати в себе алгоритми зворотного поширення помилки, алгоритми оптимізації та інші методи навчання.

Нейронні мережі можуть бути адаптовані для різноманітних завдань машинного навчання, від класифікації тексту до розпізнавання образів та генерації музики. ANN можуть автоматично виявляти та використовувати важливі залежності у вхідних даних без явного програмування. Правильно налаштовані нейронні мережі можуть ефективно узагальнювати на нові дані та уникати перенавчання. Деякі архітектури нейронних мереж можуть бути легко реалізовані на графічних процесорах (GPU) або спеціалізованих пристроях для виконання паралельних обчислень.

Нейронні мережі є потужним інструментом для класифікації та аналізу даних, а їх ефективність та потенціал зростають з розвитком технологій та методів навчання.

Усі ці методи використовуються для автоматичної класифікації даних на основі зразків та попередньо навчених моделей. Кожен з них має свої переваги та недоліки, і вибір конкретного класифікатора залежить від конкретних вимог задачі та особливостей даних.

SVM є одним із найефективніших алгоритмів класифікації в машинному навчанні з-поміж розглянутих. Порівнюючи з KNN (k-найближчими сусідами), ANN (штучними нейронними мережами), лінійними класифікаторами та іншими алгоритмами, SVM може мати декілька переваг, особливо коли йдеться про використання на спеціалізованих апаратних пристроях, таких як FPGA.

SVM добре справляється з даними високої розмірності, що робить його особливо корисним для задач з великою кількістю ознак або атрибутів. У випадках, коли кількість навчальних даних обмежена, SVM може працювати добре, навіть у порівнянні з ANN, які часто вимагають великої кількості даних для ефективної навчання. SVM може використовувати ядро для перетворення вхідних даних у вищорозмірний простір, що робить його ефективним для класифікації нелінійно розділених даних [56]. SVM може використовуватися для різних видів задач, включаючи класифікацію, регресію та виявлення аномалій.

FPGA може ефективно виконувати паралельні обчислення, що дозволяє використовувати потужність SVM для швидкої класифікації в реальному часі. FPGA надає можливість програмування на апаратному рівні, що дозволяє налаштувати SVM для конкретних вимог додатку [58,65].

Отже, SVM класифікатор може бути кращим варіантом порівняно з іншими класифікаторами для застосування на FPGA, завдяки своїм властивостям ефективності, гнучкості та спроможності працювати з обмеженими ресурсами.

1.3 Відомі методи реалізації класифікаторів FPGA

Реалізація класифікаторів машинного навчання на FPGA є важливим завданням у сфері вбудованих систем, штучного інтелекту та обробки даних [49]. Використання FPGA для класифікації дозволяє отримати високу швидкодію та енергоефективність, що є критичними для багатьох застосувань у реальному часі та обмежених обсягах ресурсів. FPGA надають можливість виконувати паралельні обчислення та забезпечують високу швидкодію обробки даних. Це особливо важливо для застосувань, які вимагають обробки в реальному часі, де навіть невеликі затримки можуть бути неприпустимими. Таким чином, використання FPGA дозволяє забезпечити потрібну швидкодію для класифікації даних у реальному часі, що робить їх привабливими для широкого спектру застосувань, від медичної діагностики до систем штучного інтелекту в автономних транспортних засобах.

Методи реалізації класифікаторів на FPGA включають у себе різноманітні техніки та підходи до реалізації алгоритмів класифікації на програмованих матрицях вентильних логічних схем (FPGA) [50].

1.4 Аналіз наявних робіт з архітектурами обладнання на основі FPGA для алгоритмів CO-SVM

Було проведене вичерпне дослідження наявних робіт з архітектурами обладнання на основі FPGA для алгоритмів CO-SVM у опублікованій літературі. Оскільки не було жодної пов'язаної роботи для CO-SVM, було розширено дослідження на наявні роботи з архітектурами обладнання на основі FPGA для загального SVM. Дослідження показало, що існує багато статей про архітектури обладнання на основі FPGA для SVM, однак було обрано для дослідження деякі роботи, які є найбільш актуальними та/або найближчими до запропонованих апаратних архітектур та методик для створення CO-SVM.

У роботі [19] була запропонована апаратна архітектура паралельної обробки на основі FPGA для SVM з використанням стохастичного градієнтного спуску (SGD) як методу навчання. Було представлено масштабованість підходу SGD для SVM щодо обчислень у фіксованій точності порівняно з одиночною точністю з плаваючою комою. Апаратний дизайн був створений за допомогою інструменту проектування Xilinx System Generator та виконаний на платі Xilinx ML605 з FPGA Virtex-6. Крім того, в апаратному дизайні не була запропонована фаза тестування. У цьому випадку були отримані та опубліковані результати синтезу щодо площі, часу та пропускної здатності; однак результати точності класифікації не були опубліковані. З отриманих результатів видно, що паралелізація призвела до збільшення зайнятої площі, що підтверджує, що вищий приріст швидкості завдяки паралелізації досягається ціною збільшення зайнятої площі на чіпі. Запропонований дизайн не міг виконувати набори даних з більш ніж 4 ознаками/атрибутами, що дійсно є обмеженням при виконанні великого обсягу даних з багатьма атрибутами.

У роботі [20] була запропонована та реалізована енергоефективна вбудована архітектура бінаризованого SVM на FPGA. Обчислювальні ядра були розроблені на мовах програмування C/C++ та перетворені в HDL з використанням інструментів високорівневого синтезу Xilinx HLS. Апаратний дизайн був виконаний на FPGA Xilinx Virtex-6/7. Результати були отримані та представлені щодо площі, швидкості, потужності та точності класифікації. Результати щодо показників продуктивності FPGA (особливо швидкості та потужності)

порівнювалися з показниками процесора та графічного процесора (GPU). З отриманих результатів видно, що FPGA та GPU досягли значного приросту швидкості в порівнянні з CPU. Однак споживання енергії GPU було значно вищим, ніж у FPGA. Ці результати показують, що апаратні архітектури на основі FPGA для SVM можуть досягати кращої продуктивності, тому вони підходять для вбудованих пристроїв з жорсткими вимогами до енергоспоживання. Було запропоновано тільки апаратний дизайн для фази тестування; однак, для підтримки реальних застосувань машинного навчання в реальному часі, необхідно надати апаратні дизайни як для фази тренування, так і для фази тестування.

У роботі [21] був запропонований апаратний прискорювач на основі FPGA для SVM, використовуючи дві техніки наближення, включаючи масштабування точності та перфорацію циклу. Апаратний засіб був розроблений за допомогою інструменту Xilinx Vivado HLS та виконаний на платі Xilinx Zynq7 ZC706. Результати були отримані та представлені щодо площі, швидкості та точності класифікації. З отриманих результатів видно, що наближене обчислення призвело до більшого приросту швидкості, але ціною більшої зайнятості площі (або використання ресурсів) на чипі та меншої точності класифікації. У деяких випадках значна втрата точності не компенсувалася значним збільшенням швидкості. Оскільки алгоритми та методи машинного навчання використовуються в багатьох додатках, де важлива точність класифікації та важливо розглядати певні пороги точності класифікації для наближення. Крім того, було вказано, що використання HLS призвело до створення апаратних архітектур, які вимагають більшої кількості ресурсів. Не було показано, як були отримані значення швидкості, оскільки не було представлено експериментальну установку для отримання часу виконання для наближеного SVM, щоб обчислити швидкість. Числова точність відіграє важливу роль у навчанні SVM з використанням оптимізації.

У [22] було запропоновано апаратне проектування на основі FPGA для класифікатора SVM. У цьому випадку були реалізовані три моделі SVM змінного розміру з використанням різних методів оптимізації. Запропонований апаратний засіб було розроблено з використанням інструменту Xilinx Vivado HLS і виконано

на платі Xilinx Zynq7 ZC702. Результати були отримані і представлені щодо площі, прискорення, потужності та точності класифікації. Як показано в [23], інструменти HLS, використані для перетворення проектів, написаних на C/C++, у HDL, не завжди виробляють найефективніше апаратне забезпечення, що може бути проблемою при реалізації цих дизайнів на ресурсозберігаючих вбудованих пристроях. Крім того, у цій статті фаза навчання проводилася автономно за допомогою програмного забезпечення; отже, опорні вектори були попередньо обчислені і передані до запропонованого апаратного засобу, який створений лише для фази тестування. Це може стати проблемою для додатків машинного навчання в реальному часі.

У [24] було запропоновано апаратне паралельне оброблення на основі FPGA для фази навчання SVM з використанням методу послідовної мінімізації (SMO). Запропонований апаратний дизайн було виконано на FPGA Xilinx Virtex-6/7/Ultra-scale. Результати синтезу були отримані і представлені щодо площі, пропускної здатності та прискорення. Однак результати точності класифікації не були повідомлені. У цьому випадку було використано ядро, оптимізоване для апаратного забезпечення (HFK) для навчання SVM, що призвело до зменшення точності числових операцій з плаваючою комою. Це, в свою чергу, зменшує точність класифікації, яка залежить від числової точності. Хоча невеликі втрати в точності прийнятні для тестування, використання HFK для навчання призведе до неефективної побудови гіперплощини під час навчання. Крім того, максимальні робочі частоти дизайнів здаються досить низькими, враховуючи, що використані FPGA можуть працювати на частоті 100 МГц або більше. З результатів та аналізу видно, що час, прискорення та пропускна здатність оцінювалися теоретично. Крім того, не було представлено жодних деталей щодо апаратних дизайнів для обчислення експонент, управління пам'яттю та умов межі прийняття рішень. Також не було запропоновано жодної фази тестування на апаратному рівні.

У статті [23] був запропонований апаратно-програмний дизайн на основі FPGA для прискорення алгоритму SVM шляхом використання дворівневого підходу: спочатку оптимізувати глобальну структуру SVM, а потім

вдосконалювати її через дослідження дизайну. Запропонована архітектура була розроблена з використанням інструменту Xilinx Vivado HLS та виконана на Xilinx Zynq Zedboard. Результати були отримані щодо площі, часу затримки та прискорення. Однак, результатів точності класифікації не було повідомлено. Як було зазначено, для високих значень параметрів SVM, використання ресурсів (тобто зайнята площа) збільшується значно, що може стати проблемою для вбудованих пристроїв з обмеженими вимогами до площі. Також було проаналізовано переваги та недоліки використання інструментів HLS для перетворення дизайнів, написаних у мові C/C+, у HDL, надаючи таким чином уявлення про неефективність HLS, що буде дуже корисним при створенні оптимізованих апаратних архітектур для поліпшення певних показників продуктивності, зокрема, часу затримки. Крім того, у апаратній частині не запропоновано навчання.

У статті [25] була запропонована апаратна архітектура на основі грубозернистої переконфігуровувальної FPGA для різних алгоритмів машинного навчання, включаючи SVM, дерева рішень та штучні нейронні мережі. Апаратура була розроблена з використанням інструменту Xilinx Vivado і виконана на FPGA Xilinx Virtex-7. Результати були отримані і представлені у вигляді площі та прискорення; однак результати точності класифікації не були зазначені. Метою даної статті було надання єдиного універсального рішення для зазначених вище алгоритмів машинного навчання. У цьому випадку для переходу від одного алгоритму до іншого було зазначено, що переконфігуровувані вузли обробки (RPNs) запропонованої архітектури можуть бути переконфігуровані індивідуально. Проте, в роботі не надається жодних деталей щодо того, як це можна зробити. Це потребує часткової переконфігурації FPGA, що додає значну складність до процесу проектування, про що не було зазначено або обговорено в статті. Крім того, з експериментами та аналізом, лише один алгоритм машинного навчання (або екземпляр) був реалізований одночасно. Для реалізації іншого алгоритму машинного навчання потрібно переконфігурувати RPN, що призводить до затрат часу на переконфігурацію. Ці затрати часу на переконфігурацію не враховуються в

прискоренні. Крім того, запропонована апаратна архітектура призначена лише для тестувальної фази.

У роботі [26] була запропонована масштабована архітектура на основі FPGA для прискорення класифікації SVM. Апаратне забезпечення було розроблено на VHDL та виконано на платі Altera Stratix III EP3SE260.

Результати були отримані та представлені щодо прискорення; однак, зайнята площа не була звільнена. Крім того, докладного обговорення та аналізу результатів точності класифікації не надано. У цій роботі було запропоновано лише апаратне забезпечення для фази тестування. Таким чином, опорні вектори були попередньо обчислені та збережені у вбудованій пам'яті для подальшої обробки під час фази тестування.

1.5 Постановка задачі

У процесі дослідження було здійснено теоретичний аналіз методів реалізації класифікаторів машинного навчання.

Було розглянуто відомі методи реалізації класифікаторів машинного навчання, а також їх переваги та недоліки.

Для реалізації швидкої та потужної системи аналізу даних необхідним є розробка та оптимізація ефективного методу інтеграції SVM-класифікатора на базі FPGA.

Поставлена мета досягається розв’язанням таких основних задач:

1. Дослідження застосування та реалізації FPGA.
2. Дослідження відомих класифікаторів машинного навчання.
3. Аналіз існуючих методів реалізації класифікаторів.
4. Розробка оптимального методу створення SVM-класифікатора FPGA.
5. Реалізація та тестування розробленого методу на FPGA.
6. Оцінка ефективності та порівняння з існуючими підходами.

1.6 Висновки

Досліджено можливості використання FPGA для аналізу даних. Було виявлено, що FPGA може ефективно використовуватися для реалізації алгоритмів машинного навчання, зокрема класифікаторів.

Проаналізовано різні класифікатори машинного навчання, їх переваги та недоліки. Зокрема, в результаті проведеного аналізу виявлено, що існуючі підходи до реалізації класифікаторів на базі FPGA мають такі недоліки, як обмежена швидкодія, обмежені можливості обробки складних алгоритмів та обмежені ресурси пристрою. Крім того, деякі підходи можуть бути недостатньо оптимізованими під конкретні завдання або дати менш точні результати через обмежену можливість роботи з нелінійними даними.

Таким чином, при повній відсутності подібної роботи в опублікованій літературі, яка б надала апаратні архітектури на основі FPGA, особливо для SVM методу основаного на опуклій оптимізації на вбудованих пристроях, або подібної роботи, яка пропонує архітектури на рівні системи, що є невід'ємною для застосувань машинного навчання для сценаріїв реального світу, виявлено потребу в розвитку оптимізованого та удосконаленого методу створення класифікатора на базі FPGA, який би ефективно використовував можливості цього пристрою та забезпечував високу точність результатів.

2 ОПУКЛА ОПТИМІЗАЦІЯ ТА МЕТОД ОПОРНИХ ВЕКТОРІВ

2.1 Оптимальна гіперплощина

З метою вирішення поставлених задач, було розглянуто основні аспекти SVM.

Концепція оптимальної гіперплощини є фундаментальною для SVM. Максимізуючи відступ, SVM має на меті знайти гіперплощину, яка не тільки розділяє класи, але і надає зону буфера для мінімізації ризику помилкової класифікації.

Математично пошук оптимальної гіперплощини передбачає вирішення задачі оптимізації, зазвичай формульованої як задача оптимізації з опуклими умовами в контексті SVM. Мета полягає в тому, щоб знайти параметри гіперплощини (ваги та зсув), які мінімізують певну функцію втрат, забезпечуючи деякі обмеження, такі як правильна класифікація навчальних даних і максимізація відступу.

За допомогою методів опуклої оптимізації SVM знаходить параметри гіперплощини, які призводять до оптимального розділення класів, що робить цей метод потужним інструментом для задач бінарної класифікації.

Оптимальна гіперплощина у методі опорних векторів (SVM) є ключовим елементом, оскільки вона визначає границю розділення між класами даних. Для бінарної класифікації, SVM шукає гіперплощину, яка максимізує відстань (відступ) між гіперплощиною та найближчими точками даних з обох класів. Це робить класифікатор більш універсальним і здатним до точної класифікації нових даних.

Оптимізація параметрів гіперплощини (ваги і зсув) зазвичай виконується шляхом розв'язання задачі оптимізації з опуклими умовами. Це означає, що шукані параметри гіперплощини знаходяться в області, де всі умови оптимізації є лінійними або квадратичними. Такий підхід гарантує знаходження глобальної мінімальної (або максимальної) точки функції втрат.

Оптимальна гіперплощина в SVM є результатом збалансованої міжкласової роздільної поверхні, яка максимізує відступ між класами і мінімізує ризик

помилкової класифікації. Це забезпечує ефективне та точне розділення класів, що робить SVM популярним інструментом для класифікації та регресії в машинному навчанні.

SVM - це загально вживаний метод класифікації, який зустрічається в багатьох різних галузях, таких як рівняння цифрових каналів у сигнальній обробці, прогнозування структури білка та виявлення раку у медичній діагностиці [15]. У цьому випадку ідея полягає в тому, щоб сформулювати гіперплощину з максимальною шириною проміжку, щоб відрізняти між двома класами [10]. Це супервізований метод класифікації, який часто включає тренувальний набір $\{A_x, B_x\}$, де - A це набір вхідних зразків/векторів даних, i - загальна кількість зразків, а B - вихідна мітка бінарного класифікатора, який використовується для ідентифікації класу вихідного зразка даних. Це представлено в рівнянні (1).

$$A = \{A_1, A_2, A_3 \dots A_x\} \forall x, A_x \in S^n$$
$$B_x \in \{-1, +1\} \quad (1)$$

Кожний вхідний вектор A складається з різних ознак набору даних, представлених як $A_x = \{A_{x_1}, A_{x_2}, A_{x_3} \dots A_{x_n}\}$, де n - кількість ознак. Формула для гіперплощини та функція рішення для визначення класу представлені відповідно рівняннями (2) та (3).

$$B_x\ (v.A + z) - 1\ \geq 0 \quad (2)$$

де v - вектор ваги, а z - значення зсуву.

$$f(A) = sgn(v.A + z) \quad (3)$$

Ширина проміжку гіперплощини може бути отримана шляхом проекції одиничного нормального вектора v на оптимальну гіперплощину, як у рівнянні (4) [17].

$$\text{Ширина проміжку} = \frac{2}{||v||} \quad (4)$$

SVM зазвичай може бути розширений до класифікації з кількома класами [22], в якій вихідна мітка представлена як $B_x \in \{B_1, B_2, B_3, \dots B_m\}$, де m - загальна кількість класів. Процес тренування для кількокласової класифікації може бути виконаний за допомогою підходу один до решти [25]. За цим підходом, m різних класів тренуються незалежно, де один набір вхідних векторів формує позитивний клас, а решта вхідних векторів формують від'ємний клас на гіперплощині. Процес класифікації може бути проведений далі, аналогічно бінарній класифікації.

2.2 Нелінійна оптимізація

Метод нелінійної оптимізації - це алгоритм або процедура, яка використовується для знаходження локального або глобального мінімуму (або максимуму) нелінійної цільової функції. Ці методи використовуються у широкому спектрі застосувань, включаючи наукові дослідження, інженерію, фінанси, машинне навчання та інші галузі.

Основні завдання нелінійної оптимізації включають пошук мінімумів або максимумів цільової функції при умовах, враховуючи нелінійні обмеження. Для цього використовуються різноманітні методи, такі як градієнтні методи, методи Ньютона, методи квазі-Ньютона, методи оптимізації з обмеженнями та інші.

У контексті машинного навчання, нелінійна оптимізація використовується для налаштування параметрів моделей на основі даних. Наприклад, у задачах класифікації та регресії нелінійні методи оптимізації можуть використовуватися для настройки вагових коефіцієнтів моделей, щоб забезпечити найкращі прогнози на основі вхідних даних.

Для покращення загальної точності завдань класифікації ширина проміжку гіперплощини може бути максимізована за допомогою нелінійного методу

оптимізації [30]. За допомогою цих методів може бути отримана цільова функція, що піддається граничним обмеженням гіперплощини, яка може бути отримана з рівнянь (2) та (4). Формула для цільової функції представлена в рівнянні (5) таким чином:

$$\underset{v}{max}\ (\text{проміжок}) = \underset{v}{max}\ (\frac{2}{||v||}) \tag{5.1}$$

$$\text{Цільова функція: } \underset{v}{min}\frac{1}{2}(|v^2|) \tag{5.2}$$

$$\text{За умови: } B_x\ (v.A + z) - 1 \geq 0\ \forall x \tag{5.3}$$

Дуальність - це концепція в оптимізації, яка виникає при розгляді проблеми оптимізації з різних точок зору. У контексті методу опорних векторів (SVM) дуальність використовується для того, щоб перевести проблему оптимізації зі змінними обмеженнями на проблему зі змінними цільової функції. Це робить можливим використання більш ефективних методів оптимізації та дозволяє отримати більш швидку збіжність.

У дуальній формі методу SVM зазвичай розглядається завдання максимізації мінімального відступу між гіперплощиною та найближчими до неї точками даних, які є опорними векторами. Це може бути виражено через знаходження мінімуму функції, яка залежить від ваги і коефіцієнтів Лагранжа, що відповідають обмеженням. Дуальний метод вирішення цієї проблеми дозволяє ефективно знаходити оптимальні значення цих коефіцієнтів і, таким чином, знаходити оптимальну гіперплощину, що розділяє дані.

У дуальній формі SVM зазвичай використовується метод Лагранжа для пошуку оптимальних значень коефіцієнтів, які максимізують мінімальний відступ між класами. Цей метод дозволяє розглядати проблему з іншого ракурсу, що може спростити її вирішення та полегшити обчислення. У дуальній формі SVM також можна зрозуміти, що максимізація мінімального відступу між класами еквівалентна мінімізації функціоналу витрат, який залежить від ваг та коефіцієнтів

регуляризації. Це дозволяє використовувати різні методи оптимізації для знаходження оптимальних параметрів моделі.

Дуальна форма SVM також дозволяє використовувати ядро, що дозволяє розширити простір ознак для кращого розділення класів у випадку, коли вони нелінійно роздільні у вихідному просторі. Це дає можливість розв'язувати більш складні задачі класифікації та регресії, використовуючи ядерні методи.

Крім того, в дуальній формі можна легко врахувати обмеження, такі як регуляризація параметрів моделі та робочі обмеження, що дозволяє налаштовувати модель з урахуванням різних вимог та умов.

Застосовуючи теорію дуальності та множники Лагранжа, можна ефективно обчислити локальні максимуми або мінімуми цільової функції в рівнянні (5). З рівняння (5) обмежена задача оптимізації може бути сформульована аналогічно нелінійному програмуванню/оптимізації. Ця обмежена задача оптимізації може бути представлена у вигляді первинної та двоїстої форм, як у рівняннях (6) та (8) [19].

$$\min L(v,z) = \frac{1}{2}v^2 - \sum_{x=1}^{m} \alpha_x \left(B_x(v.A_x + z) - 1\right) \tag{6}$$

Для отримання мінімуму з рівняння (6), розглянемо точку, де градієнт дорівнює нулю. Мінімальне значення для первинної форми може бути отримано за допомогою часткових похідних по відношенню до v та z для отримання наступних формул у рівнянні (7).

$$\sum_{x=1}^{m} \alpha_x B_x = 0 \tag{7.1}$$

$$v = \sum_{x=1}^{m} \alpha_x B_x A_x \tag{7.2}$$

Теорія дуальності корисна для обмеженої задачі оптимізації, оскільки вона надає зручний спосіб покращення класифікації даних за допомогою нелінійного підходу до оптимізації [30].

Первинна форма (у рівнянні (6)) може бути розв'язана за допомогою кількох методів, включаючи метод Ньютона, алгоритми найменших квадратів, стохастичний метод субградієнта, метод внутрішньої точки. Дуальна форма (у Рівнянні (8)) також може бути розв'язана за допомогою методів декомпозиції та методу внутрішньої точки.

$$\max V(\alpha) = \sum_{x=1}^{m} \alpha_x - \frac{1}{2}\sum_{x=1}^{m} \alpha_x \alpha_y B_x B_y \,.\, K\left(A_x, A_y\right) \qquad (8)$$

Дуальна форма має перевагу використання математичних ядер. Математичні ядра - це набір алгебраїчних функцій перетворення, які надають інформацію про подібність між ознаками даних [15]. У методі опорних векторів (SVM) математичні ядра використовуються для перетворення вхідних даних у простори вищої розмірності, де їх структури можуть бути краще розділені.

Математичні ядра в SVM дозволяють здійснювати складні нелінійні відображення даних, що може бути корисним для різноманітних задач машинного навчання. Деякі ядра можуть бути більш ефективними для певних типів даних чи задач, тому важливо враховувати характеристики конкретної задачі при виборі математичного ядра.

Для використання математичних ядер дуальна форма залежить від пар зразків, таких як $K(A_x . A_y)$, як у Рівнянні (8). Використовуючи математичні ядра, класифікація SVM може бути розширена на нелінійно роздільні набори даних.

Після отримання локальних максимумів або мінімумів з Рівнянь (6) та (8), оптимальне значення для α оцінюється для ідентифікації опорних векторів. Опорні вектори - це вхідні вектори, найближчі до гіперплощини, і мають значення α більше нуля ($\alpha > 0$). Вимірні координати опорних векторів визначають орієнтацію гіперплощини. Будь-який інший вектор може призвести до значення α, близького до нуля, що свідчить про те, що точки даних мають менший вплив на орієнтацію гіперплощини і також знаходяться далі від гіперплощини [16].

2.3 Опукла оптимізація

Опукла оптимізація - це галузь математики, що вивчає оптимізацію (мінімізацію або максимізацію) опуклих функцій на опуклих множинах. Опуклі функції мають важливі властивості, що дозволяє знаходити їхні мінімуми або максимуми ефективними методами.

У контексті машинного навчання, опукла оптимізація використовується для розв'язання багатьох задач, таких як навчання моделей машинного навчання, включаючи метод опорних векторів (SVM), лінійну регресію, логістичну регресію та багато інших.

Методи опуклої оптимізації дозволяють знаходити оптимальні рішення для цих завдань, забезпечуючи ефективність та гарантовану збіжність до оптимуму. Вони є ключовим інструментом у сфері машинного навчання для навчання моделей на великих обсягах даних з великою кількістю параметрів.

Опукла оптимізація є особливо важливою для методів машинного навчання, оскільки вона забезпечує гарантію збіжності до оптимального рішення. Основна ідея полягає в тому, щоб знайти такий набір параметрів моделі, який мінімізує (або максимізує) величину втрат або функцію витрат на навчальних даних, при цьому дотримуючись обмежень на параметри моделі.

Одним із найбільш відомих прикладів використання опуклої оптимізації у машинному навчанні є метод опорних векторів (SVM), який використовується для класифікації та регресії. У цьому методі, завдання класифікації редукується до пошуку оптимальної гіперплощини, яка найкращим чином розділяє дані різних класів. Цей пошук гіперплощини формулюється як задача опуклої оптимізації.

Застосування опуклої оптимізації у машинному навчанні дозволяє знаходити оптимальні рішення на великих обсягах даних, забезпечуючи ефективність та стабільність процесу навчання.

Зазвичай для великомасштабних застосувань, через великий обсяг даних, обмежена задача оптимізації, представлена в Рівнянні (8), повинна збігатися до мінімального значення, щоб знайти найкраще призначення для побудови

оптимальної гіперплощини. Для опуклої оптимізації всі локальні мінімуми вважаються глобальним мінімумом [19]. У цьому випадку, через наявність шуму в деяких наборах даних, параметри м'якого проміжку, такі як параметр помилки l-норми (параметр ціни W та змінна слабкості (τ)), повинні бути враховані для кращої узагальненості первинної форми. Оскільки згадана нелінійна оптимізація складається з обмеження рівності, дуальна форма (у Рівнянні (8)) може бути зведена до загальної форми задачі опуклої оптимізації, інтегруючи параметри м'якого розділення. Таким чином, загальна цільова функція у Рівнянні (8) може бути змінена до наступного Рівняння (9).

Дуальна форма, цільова функція:

$$\min_{\alpha} V(\alpha) = \frac{1}{2}\sum_{x,y=1}^{m} \alpha_x N \alpha_y - \sum_{x=1}^{m} u^Q \alpha_x \tag{9.1}$$

$$\text{При умові: } 0 \leq \alpha_x \leq W \tag{9.2}$$

$$\sum_{x=1}^{m} \alpha_x B_y = 0 \tag{9.3}$$

Рівняння (9) схоже на загальну форму опуклої оптимізації, з обмеженням рівності, і може бути записане як рівняння (10) нижче.

Дуальна форма, цільова функція:

$$\min_{A} f(A) = \min_{A} \frac{1}{2} A^Q NA - u^Q A \tag{10.1}$$

$$\text{При умові: } 0 \leq GA_x \leq h \tag{10.2}$$

$$D.A = z \tag{10.3}$$

У опуклій оптимізації вказана цільова функція (у Рівняннях (9) та (10)) повинна збігатися таким чином, щоб вона задовольняла умову матриці Гессе [30], яка стверджує, що контур опуклої площини повинен бути неперервно диференційований, як у Рівнянні (11). Потім оптимальне рішення можна знайти там, де значення градієнта дорівнює нулю.

$$\frac{\delta B}{\delta A} * \frac{\delta B}{\delta A} \geq 0 \quad (11)$$

У Рівнянні (10) матриця P є симетричною та позитивно напіввизначеною; як цільова функція, так і функція обмежень є опуклими. Особливо для великомасштабних застосувань отримані матриці є щільними, тому важкі для розв'язання. Тому часто використовуються методи декомпозиції для розбиття процесу оптимізації за заданою умовою знаходження двох робочих наборів [25,26], як показано у Рівнянні (12). Правильний вибір "робочих наборів" впливає на продуктивність алгоритму оптимізації та його властивостей збіжності. У цьому випадку процес знаходження двох робочих наборів для визначення опорних векторів у тренувальному наборі є обчислювально-інтенсивним та ітеративним процесом. Отже, надаючи спеціалізовані та оптимізовані акселератори на базі FPGA, можна значно покращити швидкість та продуктивність таких обчислювально-інтенсивних застосувань (або завдань) [47,48,52].

Методи декомпозиції, такі як послідовна мінімальна оптимізація, використовуються для вирішення завдань нелінійної оптимізації шляхом послідовного вибору робочого набору на основі близькості до цільової функції (у Рівнянні (10)). Враховуючи різноманітні методи вирішення нелінійної оптимізації, послідовна мінімальна оптимізація (SMO) є найпопулярнішою, завдяки її здатності ефективно обробляти великомасштабні набори даних. У цьому випадку на кожній ітерації вхідний вектор A (у Рівнянні (10)) розділяється на два робочих набори як $(A_k, \overline{A_k})$, де k - поточний лічильник ітерацій, A_k - поточний вхідний вектор, а $\overline{A_k}$ - попередній вхідний вектор. На основі вказаної початкової точки об'єктивна функція (у Рівнянні (10)) розв'язується для збіжності до мінімального значення [17].

Для опуклої оптимізації (з Рівняння (10)) початковою допустимою точкою є (0,0), що є першим вхідним зразком даних у Рівнянні (1). Потім, на основі напрямку спуску градієнта вибирається наступний робочий набір. Вибір допустимої точки та робочого набору може впливати на загальний час, необхідний для вирішення

оптимізації квадратичної оптимізації. Рішення для опуклої оптимізації (у Рівнянні (10)) знаходиться шляхом вибору відповідного робочого набору, який задовольняє наступні критерії у Рівнянні (12).

$$\max_{x\epsilon S(\alpha^*)}\left\{-\frac{\nabla f(\alpha^*)}{B_y}\right\} \leq \min_{x\epsilon O(\alpha^*)}\left\{-\frac{\nabla f(\alpha^*)}{B_y}\right\} \tag{12}$$

У Рівнянні (12) S(α) та O(α) - це множини індексів, які використовуються для характеризації напрямку спуску та визначення умов оптимальності. На основі напрямку спуску об'єктивна функція (у Рівнянні (9)) збігається до мінімального значення α, яке використовується для визначення орієнтації розділяючої гіперплощини. У цьому випадку побудова гіперплощини створює явну різницю між різними класами даних.

Нелінійні методи оптимізації для SVM, використовувані для побудови гіперплощини, значно покращують загальну швидкість та продуктивність класифікатора SVM і також призводять до високої точності результатів [78]. Однак, отримання мінімальних значень об'єктивної функції (у Рівняннях (9) та (10)) є обчислювально-інтенсивним завданням, яке залежить від кількості вхідних вибірок. Тому надзвичайно важливо розробити та надати спеціалізовані та оптимізовані прискорювачі на базі FPGA для покращення прискорення цих обчислювально-інтенсивних завдань.

2.4 Висновки

Досліджено та проаналізовано важливі аспекти оптимальної гіперплощини, яка є центральним елементом у методі опорних векторів. Виявлено, що оптимальна гіперплощина максимізує роздільну здатність між класами, забезпечуючи найбільш ефективне розділення даних у просторі.

Під час аналізу виявлено, що правильний вибір гіперплощини дозволяє підвищити точність та надійність класифікації для різноманітних наборів даних.

Досліджено методи нелінійної оптимізації, які використовуються для побудови нелінійних розділяючих гіперплощин у випадках, коли дані не є лінійно роздільними.

Встановлено, що використання таких методів дозволяє ефективно вирішувати задачі класифікації нелінійних даних, що розширює можливості застосування методу опорних векторів на різні типи даних.

Досліджено опуклу оптимізацію як ключовий компонент методу опорних векторів. Виявлено, що використання опуклої оптимізації дозволяє ефективно знаходити глобальний мінімум функції втрат для оптимального розташування гіперплощини у просторі. Такий підхід забезпечує стійкість та надійність роботи методу опорних векторів навіть у випадку складних та нелінійних задач класифікації.

3 МЕТОД СТВОРЕННЯ SVM-КЛАСИФІКАТОРА ДЛЯ АНАЛІЗУ ДАНИХ НА БАЗІ FPGA

3.1 Основи створення SVM-класифікатора для аналізу даних на базі FPGA

З метою підвищення ефективності виконання аналізу даних та удосконалення підходу до реалізації машинного навчання шляхом інтеграції SVM-класифікатора з FPGA, було запропоновано метод інтеграції SVM-класифікатора.

Було досліджено різні класифікатори машинного навчання та вирішено зосередитися на алгоритмі класифікації SVM, оскільки у SVM є багато переваг/властивостей, що вважаються відповідними для застосувань машинного навчання. Наприклад, SVM більш підходить для класифікації нелінійно роздільних даних, дозволяє класифікацію в багатовимірному просторі, може обробляти високо-вимірні дані, та забезпечує високі результати точності. Крім того, точність та процес класифікації SVM можуть бути в подальшому вдосконалені, використовуючи нелінійні методи оптимізації для пошуку оптимальних рішень. Тому було запропоновано інтегрувати метод оптимізації опуклої функції (один з найкращих нелінійних методів оптимізації) до класифікатора SVM, щоб чітко відрізняти два окремих класи шляхом максимізації ширини маржі гіперплощини, що в свою чергу призводить до оптимального рішення [19,30].

У даній роботі обидві версії різних операцій та завдань, як апаратні, так і програмні, реалізовані з використанням ієрархічного платформного та модульного підходів до проектування для спрощення повторного використання компонентів на різних рівнях абстракції. Як показано на рисунку 3.1, на найвищому рівні даний алгоритм SVM на основі CO (опукла оптимізація) складається із завдань навчання та тестування. Під час процесу навчання, на основі Рівнянь (9) та (10), об'єктивна функція формулюється для визначення гіперплощини, вибору відповідного математичного ядра та отримання оптимального рішення. Під час процесу тестування, функція прийняття рішення класифікатора обчислюється з використанням операції перевірки знаку (у Рівнянні (3)). Зазначені проміжні операції для обробки завдань навчання та тестування, включаючи

додавання/множення векторів, обчислення матриць та різноманітні арифметичні операції, які розміщені на найнижчому рівні ієрархії проектування на базі платформи [42].

Рисунок 3.1 – Підхід до проектування на основі ієрархічності та модульності

3.2 Підхід до проектування та платформа розробки DE1-SoC

Всі дослідження з вбудованим обладнанням та програмним забезпеченням було виконано на платформі розробки DE1-SoC Board, що використовує пристрій Altera Cyclone V SoC 5CSEMA5F31C6.

Вигляд плати для розробки DE1-SoC від Terasic представлений на рис 3.2 та рис 3.3 [34].

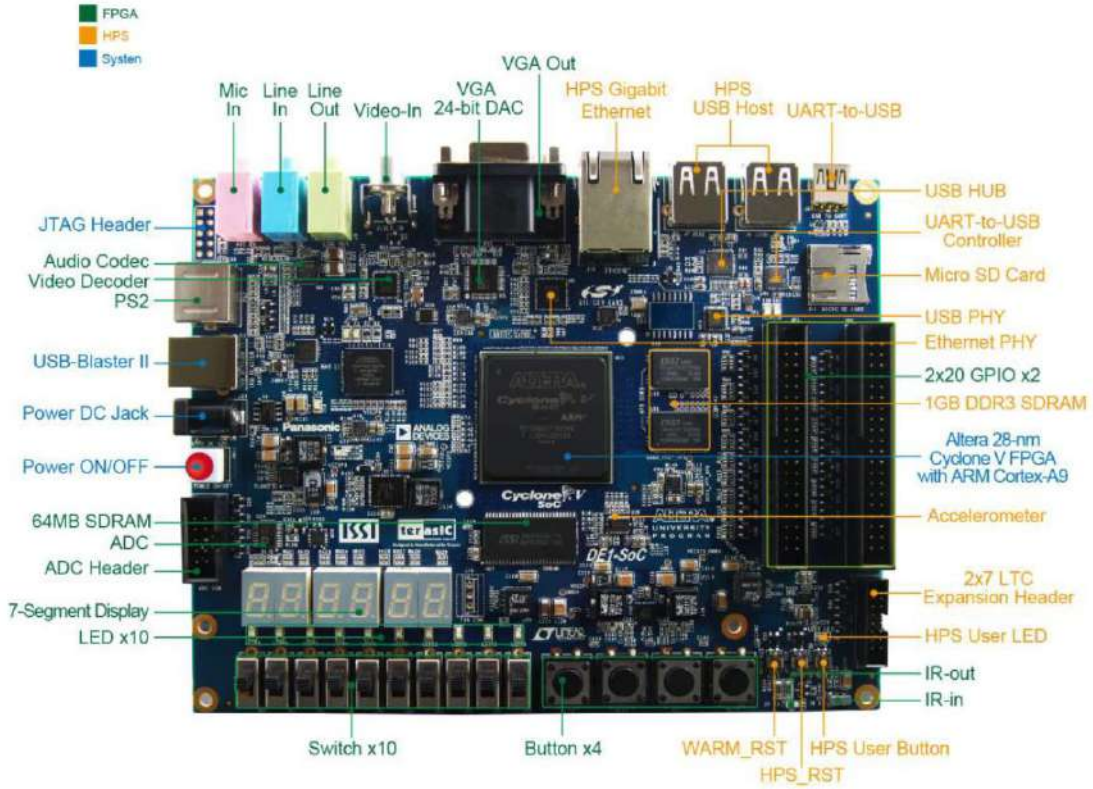


Рисунок 3.2 – Вид зверху плати для розробки DE1-SoC

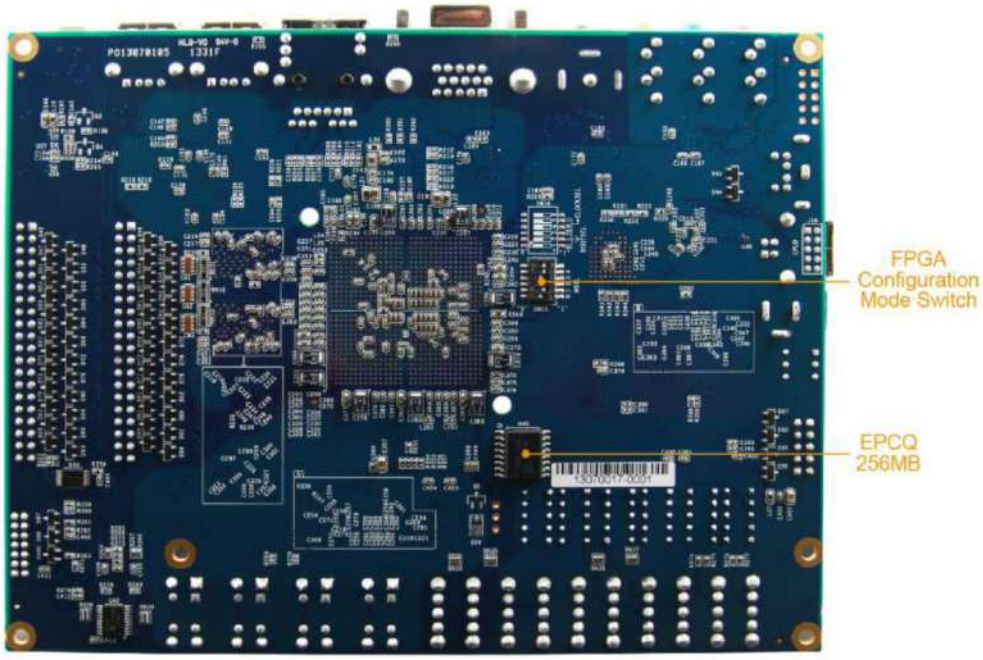


Рисунок 3.3 – Вид знизу плати для розробки DE1-SoC

Основні технічні характеристики:

- Altera Cyclone V SE 5CSEMA5F31C6N

- Пристрій для конфігурації – EPCS128
- USB-Blaster II для програмування; JTAG режим
- 64MB SDRAM (16-bit шина даних)
- Чотири тактових генератори тактових сигналів на 50МГц
- Двох-ядерний процесор ARM Cortex-A9 MPCore
- Частота роботи :800MHz
- 1GB DDR3 SDRAM (32-bit шина даних)
- 1 Gigabit Ethernet PHY з RJ45 роз'ємом

Усі запропоновані спеціалізовані вбудовані апаратні модулі розроблені з використанням мішаних мов VHDL і Verilog [44,45,46]. Вони виконуються на вказаній вище Altera Cyclone FPGA Dec1-Soc, що працює з частотою 100 МГц (в реальному часі), для перевірки їхньої правильності та продуктивності. Всі дані програмні модулі вбудованого обладнання написані на C++ та виконуються на 32-розрядному процесорі ARM Cortex-A9 з такою самою частотою 100 МГц на тій же FPGA. Швидкодія оцінюється за допомогою часу виконання базового програмного забезпечення порівняно з часом виконання покращеного апаратного забезпечення. Час виконання апаратного та програмного забезпечення отримуються з AXI Timer [68,72].

Загальний приріст швидкодії оцінюється за допомогою двох різних наборів тестових даних, отриманих з репозиторію машинного навчання UCI: датасет «Prediction of Pulsar Star» (набір даних для прогнозування зірки-пульсар) [40] та датасет «Iris Flowers» (набір даних квіти Іриси) [39] для машинного навчання.

Набір даних «Pulsar» складається з 16 259 помилкових вибірок, викликаних радіочастотними перешкодами/шумом, і 1639 реальних вибірок пульсарів. Кожен рядок спочатку містить перелік змінних, а мітка класу є останнім записом. Використовувані мітки класу: 0 (негативний) і 1 (позитивний).

Цей набір даних стосується ідентифікації зірок-пульсарів, які є рідкісним типом нейтронних зірок, що випромінюють радіохвилі, які можна виявити на

Землі. Пульсари цікаві для наукових досліджень, оскільки вони можуть надати розуміння простору-часу, міжзоряного середовища та станів матерії.

У цьому наборі даних метою є використання алгоритмів класифікації для розрізнення двох типів прикладів: реальних пульсарів (позитивний клас) і фальшивих сигналів, викликаних перешкодами або шумом (негативний клас).

Набір даних містить інформацію про вісім різних характеристик для кожного кандидата, як-от статистичні дані з пульсового профілю та кривої DM-SNR, і використовує ці атрибути, щоб класифікувати, чи є кандидат пульсаром (1) чи ні (0).

Датасет «Iris Flowers» (Іриси Фішера) - це багатовимірний набір даних для задачі класифікації. Іриси Фішера складаються з даних про 150 вимірювань ірисів з трьох видів — Iris setosa, Iris virginica і Iris versicolor, по 50 вимірювань на вид. Для кожного екземпляра є чотири характеристики (в сантиметрах):

- довжина зовнішньої частини чашолистка (пелюстки);
- ширина зовнішньої частини чашолистка (пелюстки);
- довжина внутрішньої частини чашолистка (пелюстки);
- ширина внутрішньої частини чашолистка (пелюстки).

За даними вимірами будують правила класифікації, що дозволяє визначити вид рослини.

3.3 Архітектура на рівні системи

На рисунку 3.4 показана системна архітектура для вбудованих апаратних та програмних проектів. Оскільки 2 МБ вбудованої пам'яті BRAM на мікросхемі Altera Cyclone V FPGA на платі Dec1-Soc недостатньо для зберігання великої кількості даних, які зазвичай зустрічаються в багатьох додатках машинного навчання, було інтегровано зовнішню пам'ять DDR3-SDRAM об'ємом 512 МБ в систему [73,74]. У цьому випадку DDR3-SDRAM та контролер пам'яті DDR3-SDRAM працюють на частоті 200 МГц, тоді як інша частина системи працює на частоті 100 МГц [43]. Як показано на рисунку 3.2, було використтано шину AXI

(Advanced Extensible Interface) для полегшення комунікації між периферійними пристроями на рівні системи.

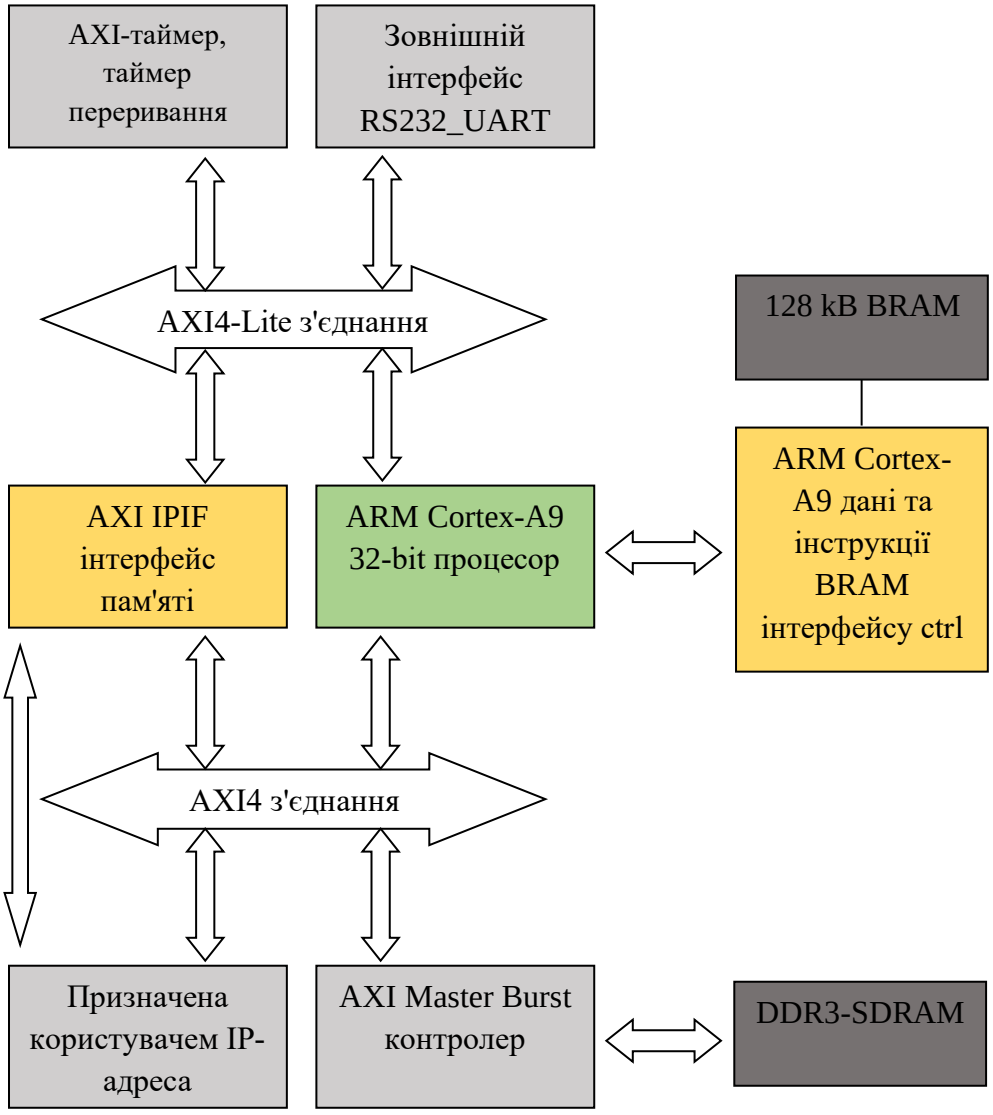


Рисунок 3.4 – Архітектура на рівні системи

Під час початкової фази проектування програмного забезпечення було сконфігуровано процесор ARM Cortex-A9 таким чином, щоб мати максимально доступну кеш-пам'ять розміром 128 КБ [51]. Однак, ці 128 КБ кеш-пам'яті недостатньо для виконання програмного коду та обробки даних. Тому було змінено розміри стеку, а також збільшено адресацію кеш-пам'яті, щоб забезпечити 256 КБ.

Як показано на Рисунку 3.4, дана користувацька Інтелектуальна Власність (IP) взаємодіє з DDR3-SDRAM та ARM Cortex-A9 за допомогою шини AXI через

модуль Інтелектуальної Власності Інтерфейсу AXI (IPIF), використовуючи набір портів, званих Інтелектуальна власність міжпідключення (IPIC). Зазвичай після того, як процесор ARM Cortex-A9 надсилає сигнал старту даному користувацькому IP через шину AXI4-lite та інтерфейс AXI IPIF, користувацький IP починає обробку, безпосередньо читає/записує дані/результати з/до DDR3-SDRAM та надсилає сигнал завершення процесору ARM Cortex-A9, коли виконання завершується.

Для даного апаратного проекту було надано інтерфейс високої пропускної здатності AXI4 burst/stream для потокового передавання даних з DDR3-SDRAM до користувацького IP для обробки в реальному часі. Одна з цілей даного проектування полягала в створенні архітектури на рівні системи таким чином, щоб тренувати і класифікувати безперервний потік даних, використовуючи інтерфейс AXI4 burst/stream. У реальному сценарії застосування ця функція дозволяє забезпечити пряме з'єднання між користувацьким апаратним IP та камерою (наприклад, в автономному автомобілі), щоб обробляти вхідні дані в режимі реального часу. Це дозволяє даному апаратному IP динамічно виконувати процеси тренування та класифікації для адаптації до постійно змінюючого середовища. Завдяки потоковому передаванню даних та їх обробці безпосередньо зменшується кількість пам'яті, необхідної для вбудованих конструкцій [27].

3.4 Процес попереднього завантаження та архітектура верхнього рівня

Під час дослідження було виявлено, що значна кількість часу витрачається на доступ до пам'яті DDR3-SDRAM поза мікросхемою, що є основним чинником обмеження продуктивності. Тому було розроблено та інтегровано метод попереднього завантаження для вирішення проблеми затримки доступу до пам'яті у визначеному користувацькому апаратному IP [35].

Як показано на Рисунку 3.4, інтерфейси AXI4-lite та AXI4 виступають як "клейова" логіка для всієї системи, включаючи процесор ARM Cortex-A9, внутрішні периферійні пристрої та визначене користувацьке IP. AXI4-lite є однократним транзакційним двонаправленим інтерфейсом, з пам'яттю,

відображеною в області пам'яті. ARM Cortex-A9 надсилає/отримує певні керуючі сигнали, а також відслідковує стан користувацького IP через шину AXI4-lite та через робочі реєстри (або реєстри, доступні за допомогою програмного забезпечення). Ці 32-бітні робочі реєстри також використовуються для надсилання специфікацій SVM, таких як тип ядра, розміри вхідних векторів, параметри ціни/слабкої величини, до визначеного користувацького IP, разом із початковою адресою пам'яті DDR3-SDRAM для доступу до наборів даних.

Окрім AXI4-lite, визначений користувацький IP зчитує/записує дані/результати з/у DDR3-SDRAM через шину AXI4. AXI4 master burst - це інтерфейс високопродуктивної пам'яті, здатний передавати пакети даних розміром до 256 бітів даних, які сумісні з шириною даних 16, 32, 64 і 128 з однофазною транзакційною фазою адреси. За такої ширини даних, можна передавати до 1 МБ (2n - 1 байтів) за цикл на інтерфейсі команд IPIC. Використовуючи можливості AXI4 master burst, час доступу до пам'яті значно зменшується для процесів тренування та тестування SVM.

Запропонований метод попереднього завантаження показаний на Рисунку 3.5. Під час режиму попереднього завантаження, модуль користувацької логіки визначає загальну кількість байтів, які потрібно завантажити, використовуючи зазначені вище специфікації SVM, надані ARM Cortex-A9 через робочі реєстри. Далі, генератор адрес та контролер AXI master burst налаштовують сигнали керування IPIC з відповідною шириною даних, довжиною вибуху та кількістю кадрів на цикл. Потім контролер AXI master burst відправляє зазначені вище деталі про дані, а також сигнал "запит на читання" до ядра AXI з'єднання. Як тільки AXI master отримує сигнал підтвердження запиту на читання від ядра AXI з'єднання, користувацький IP може почати отримувати дані з DDR3-SDRAM, може зберігати дані в BRAM. Після цього логіка користувача може розпочати процес тренування. У цьому випадку генератор адрес є необхідним для індексування правильних значень для матричних обчислень. Після завершення процесу тренування, контролер AXI master burst автоматично налаштовується для операції запису, щоб

зберегти вектори ваги та значення зсуву в DDR3-SDRAM для подальших обчислень/аналізу.

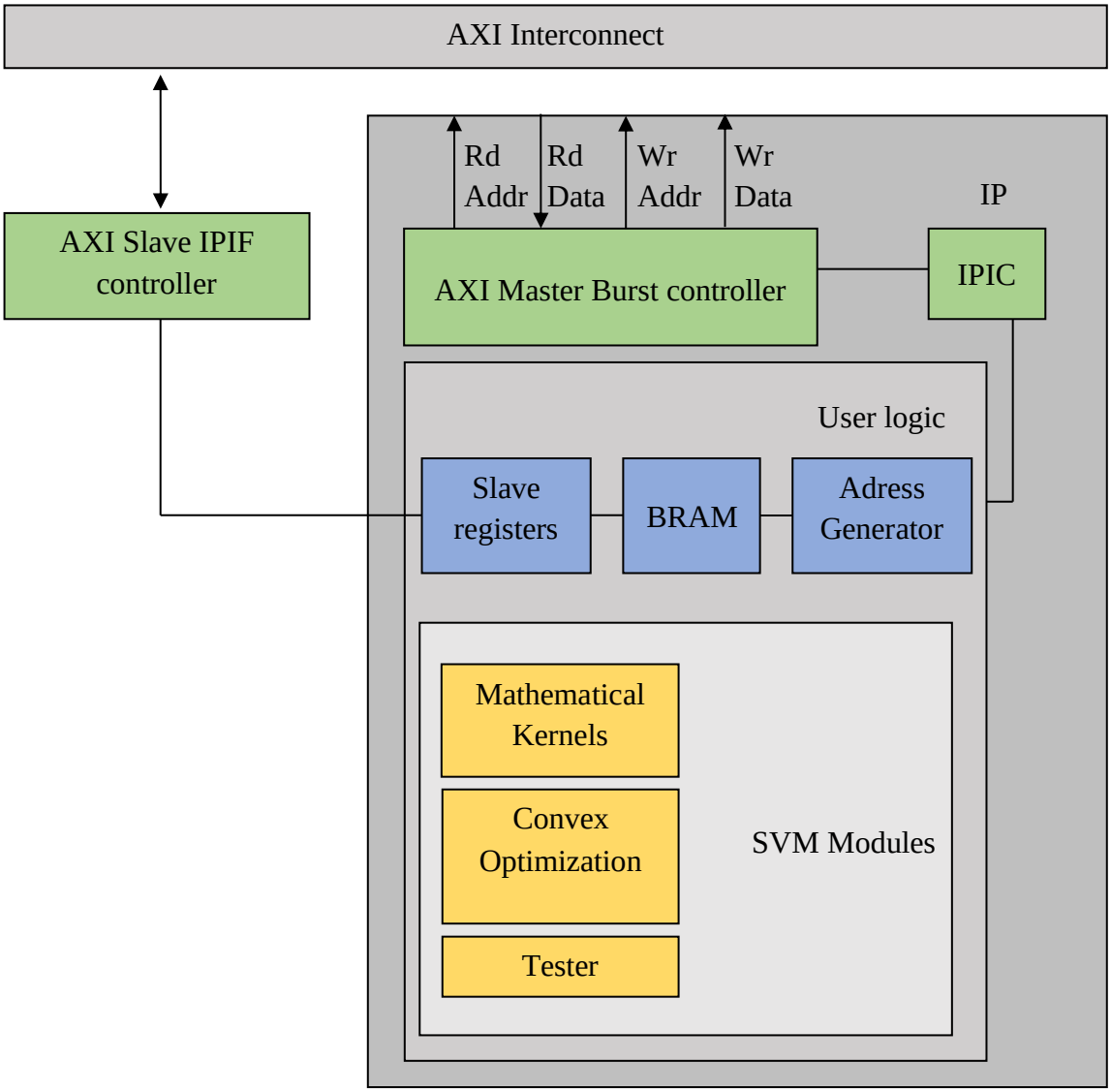


Рисунок 3.5 – Метод попереднього завантаження та верхній рівень архітектури

3.5 Проектування вбудованого програмного забезпечення для SVM на основі опуклої оптимізації

Для розробки програмного забезпечення для даного алгоритму SVM на основі опуклої оптимізації, використано C++ за допомогою інструментів розробки Microsoft Visual Studio. Цей дизайн програмного забезпечення виконується на настільному комп'ютері з процесором Intel i7, що працює на частоті 2,3 ГГц.

Отримані результати порівнюються та перевіряються з результатами відкритого коду Python, отриманого з [36]. Як результати на C++, так і на Python також використовуються для перевірки отриманих результатів (тобто для перевірки правильності функціональності та роботи, а також точності) даних вбудованих апаратних та програмних архітектур.

Щоб врахувати обмежену природу ресурсів вбудованих пристроїв, було значно модифіковано вищезгадану архітектуру програмного забезпечення на C++, спочатку розроблену для настільних комп'ютерів. У цьому випадку розробляється код більш ефективним та простішим, так, щоб вміщувався у доступну пам'ять програмного кешу вбудованого мікропроцесора, тобто ARM Cortex-A9, не впливаючи на внутрішню структуру/потік та функціональність загального алгоритму [53].

Для даного вбудованого програмного забезпечення процесор ARM Cortex-A9 налаштований на максимальну доступну пам'ять кешу розміром 128 КБ, з яких 64 КБ використовується для кешу інструкцій, а 64 КБ - для кешу даних. Є можливість змінювати потужність цих кешів даних і інструкцій, проте загальна їхня сума не може перевищувати максимальну доступну пам'ять кешу [57].

Під час фази розробки вбудованого програмного забезпечення були виявлені кілька проблем через жорсткі обмеження вбудованих пристроїв. Однією з основних проблем є обмежені ресурси пам'яті. У цьому випадку деякі функціональності звичайних програм на C++, що виконуються на настільних комп'ютерах, не можуть бути прямо розроблені та реалізовані на вбудованих пристроях. Наприклад, імпорт директиви попередньої обробки для обчислень векторів з настільного комп'ютера на вбудовані пристрої може призводити до проблем з обмеженнями пам'яті. Щоб вирішити цю проблему, було змінено розміри стеку; потім розроблено компактну функцію в програмному забезпеченні, яка здатна ефективно виконувати векторні обчислення.

3.6 Вбудований апаратний дизайн

Після аналізу функціонального потоку SVM алгоритму на основі опуклої оптимізації, було визначено, що цей складний алгоритм потрібно розбити на три етапи (SVM Module на рисунку 3.5), щоб спростити процес проектування. Операції цих трьох послідовних етапів - це математичні ядра, опукла оптимізація (або розв'язувач опуклої оптимізації) та тестування (або тестер). Варто зазначити, що розв'язувач було розглянуто як апаратний IP для виконання етапу оптимізації.

Апаратні розробки для кожного етапу включають у себе шлях даних та шлях керування. Шлях керування складається з скінченних автоматів (FSM) та керує контрольними сигналами шляху даних та BRAM. Було спроєктовано також модуль верхнього рівня (тобто SVM Module на рисунку 3.5.), щоб інтегрувати три модулі для трьох етапів.

Модуль верхнього рівня забезпечує необхідну комунікацію та керування між трьома етапами. Шлях керування модуля верхнього рівня також складається з кількох скінчених автоматів, мультиплексорів та буферів з трьох станів для керування часом, маршрутизацією та внутрішніми структурами і функціональностями проєктування.

Три етапи алгоритму SVM на основі CO виконуються не послідовно для використання паралельної обробки, характерної для апаратного забезпечення на основі FPGA [54,56]. Спочатку перший етап, тобто математичне ядро, обробляється до отримання певної кількості результатів.

Потім N та U-матриці (у рівняннях (9) та (10)) на другому етапі, тобто оптимізація (або розв'язування), обчислюються на основі результатів ядра, поки інша частина математичного ядра все ще обробляється. Зазвичай час виконання обчислення матриць N та U менший, ніж час виконання математичного ядра, і він залежить від результатів останнього.

Тому було спроєктовано та розроблено простий лічильник для введення затримки часу, щоб переконатися, що другий етап починається лише після того, як ядро обробило принаймні 50 зразків навчальних даних, що становить приблизно 10% загального розміру даних, перш ніж почати другий етап. Для всіх етапів

проміжні/кінцеві результати зберігаються в BRAM, і після обробки всіх трьох етапів кінцеві результати записуються в DDR3-SDRAM.

Ці внутрішні архітектури налаштовані та оптимізовані таким чином, що вони використовують вроджений паралелізм та конвеєрну природу алгоритму SVM на основі СО.

3.6.1 Етап 1: Математичні ядра

На першому етапі даного апаратного проєкту було обрано та виконано відповідне математичне ядро. Лінійно-нероздільні вектори в просторі вводу можуть бути перетворені в лінійно-роздільні вектори в просторі ознак шляхом відображення точок даних в більш високовимірний простір. Це перетворення можна виконати за допомогою математичних ядер (включаючи лінійні, поліноміальні та гаусівські ядра), оскільки обмеження не залежить від розмірності простору для SVM. Це ефективний спосіб отримання чітко визначеної роздільної гіперплощини. Математичні ядра - це набір алгебраїчних функцій перетворення, які забезпечують інформацію про схожість між ознаками даних. Для використання математичних ядер функція відображення (Ф(x)) в Рівнянні (13) повинна задовольняти умову Мерсера, яка стверджує, що скалярний добуток двох вхідних векторів має бути визначений для всіх ознак, як показано в Рівнянні (13).

$$K(A_x, A_y) = \Phi(A_x) \times \Phi(A_y) \tag{13.1}$$

$$\int\int g(A_x)K(A_x, A_y)g(A_y)dA_x dA_y \geq 0 \tag{13.2}$$

Було вирішено створити індивідуальні та оптимізовані вбудовані архітектури для лінійного, поліноміального та гаусового ядра з радіальною базисною функцією (RBF), оскільки це три найпопулярніші математичні ядра, що використовуються для алгоритму SVM. Рівняння для цих трьох математичних ядер представлено у Рівняннях 14, 15, 16:

Лінійне ядро:

$$K(A_x, A_y) = A_x \times A_y \tag{14}$$

Поліноміальне ядро:

$$K(A_x, A_y) = (c + A_x \times A_{xy})^s \tag{15}$$

Гаусове RBF ядро:

$$K(x_x, x_y) = e^{-\gamma}(x_x - x_y)^2 \tag{16}$$

Шлях даних для поліноміального ядра показано на Рисунку 3.6 (відповідає модулям 2), тоді як схема даних для лінійного ядра позначена пунктирними лініями на Рисунку 3.6 (відповідає модулям 1). Як показано, схема даних лінійного ядра складається з множника, суматора та накопичувального реєстру із зворотним зв'язком до суматора, тоді як схема даних для поліноміального ядра має ще один суматор та модуль степені.

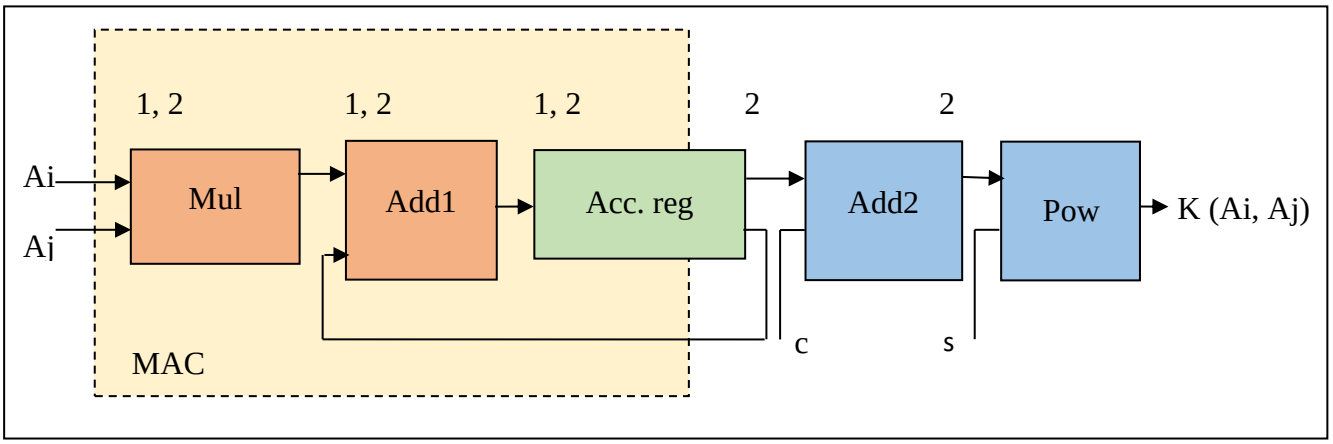


Рисунок 3.6 – Шлях даних для лінійного та поліноміального ядер

Результат лінійного ядра - це операція скалярного добутку двох вхідних вибірок даних. У цьому випадку спочатку з BRAM зчитуються перші два елементи двох вибірок даних - перші два елементи першого ряду, та проводиться операція

множення, за якою виконується накопичувальна операція для кожного результату множення. Цей процес буде продовжуватися до тих пір, поки ця операція множення та накопичення (MAC) не буде виконана для останніх двох елементів двох вибірок даних. Потім кінцевий результат операції MAC передається і зберігається на BRAM для подальшого аналізу/обчислень. Як показано на Рисунку 3.6, модулі у пунктирних лініях складаються з операції MAC.

Для шляху даних для поліноміального ядра, входами до другого суматора є кінцевий результат від лінійного ядра, а також коефіцієнт c. Цей результат додавання проходить через модуль степеня, щоб виконати "степінь s" для результатів додавання. У цьому випадку було створено апаратний модуль для функції степеня, використовуючи простий цикл для ітерації множення на основі вказаного значення степеня (s). Хоча це значення степеня параметризоване в даному проєктуванні, загально використовуване квадратичне ядро виконується, вважаючи це значення степеня як два (2). Кінцевий результат цього модуля степеня також передається і зберігається на BRAM для подальших обчислень.

Шлях даних для ядра Гаусса показаний на рисунку 3.7. Гауссівське ядро з радіальною базисною функцією (RBF) є найпопулярнішим серед згаданих трьох математичних ядер. Шлях даних складається з 3 модулів MAC для виконання частини Рівняння (16), в якому $\left(A_x - A_y\right)^2$, розширене до $A_x^2 + A_y^2 - 2A_x.A_y$, що потребує трьох операцій дотичного добутку. Далі виконується операція додавання на результати двох квадратних операцій, тоді результат операції MAC3 множиться на два. Потім результат множника віднімається від результату операції додавання. Результат віднімання множиться на параметр, відомий як бета (β). Параметр бета (β) визначає вплив вибірки даних на розділяючу гіперплощину [36]. У цьому випадку значення бета (β) зазвичай змінюється від 10^{-3} до 10^3 за потребою. В даному проєктуванні замінюється значення бета (β) від 2×10^{-3} до 2×10^2 як для апаратних, так і для програмних вбудованих конструкцій. Як показано на Рисунку 3.7, результат операції множення (з Mult2) проходить через модуль експоненти для отримання кінцевого результату ядра Гаусса з радіальною

базисною функцією (RBF). У цьому випадку створюється параметризований апаратний модуль експоненти на основі розкладу Тейлора, який представлений рівнянням (17) [37,38]. Оскільки кількість елементів в ряді Тейлора стає нескінченною, для обох апаратних та програмних вбудованих конструкцій обмежується кількість елементів першими 5 елементами ряду Тейлора, як показано в рівнянні (17), не жертвуючи точністю кінцевого результату експоненти. Далі створюється модуль експоненти таким чином, щоб обробляти кожний елемент ряду Тейлора послідовно (тобто обробляти один елемент за один раз). Шлях даних модуля експоненти складається з множника, дільника та суматора, які використовуються для обробки кожного елементу ряду Тейлора ітеративним способом. Кінцевий результат цього модуля експоненти пересилається та зберігається в BRAM для подальших обчислень.

$$e^{A} = 1 + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \frac{A^4}{4!} \tag{17}$$

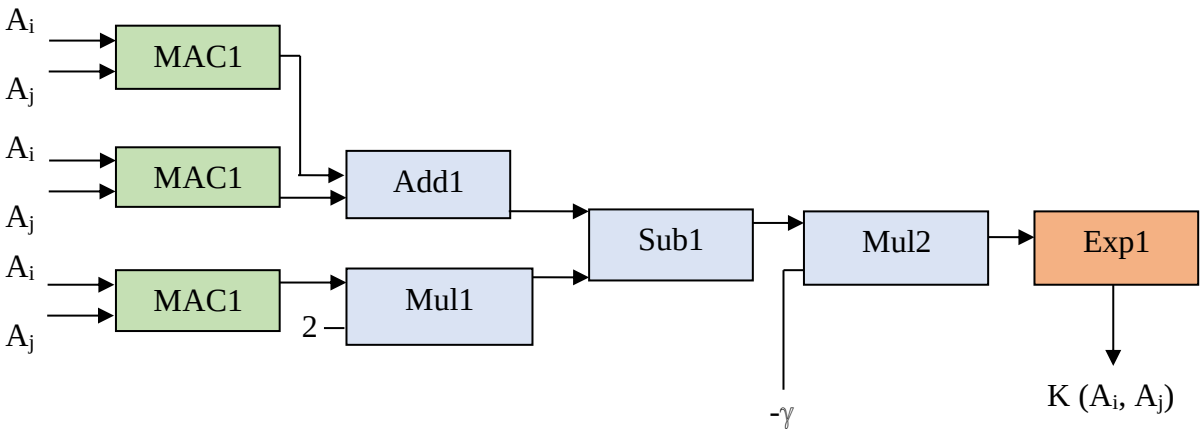


Рисунок 3.7 – Шлях даних для ядра Гауса радіальної базисної функції (RBF)

Розмір вихідної матриці для обчислення ядра залежить від кількості вхідних вибірок даних (m). Таким чином, розмір матриці $K(A_x, A_y)$ становить mAm. Після обробки 10% даних для обчислення ядра ініціюється процес оптимізації за опуклою

функцією. Обчислення для відстеження 10% обробки даних реалізоване за допомогою простого лічильника. Обчислення ядра необхідне для виконання кожної ітерації наступного етапу, що є оптимізацією за опуклою функцією. Тому спочатку результати математичного ядра зберігаються на BRAM для спрощення ітеративного процесу оптимізації за опуклою функцією.

3.6.2 Етап 2: Опукла оптимізація

Етап оптимізації є найбільш складною операцією серед трьох етапів алгоритму SVM на основі оптимізації за опуклою функцією (CO). Щоб зменшити складність, етап 2 поділяється на три фази: ініціалізація параметрів, оптимізація за опуклою функцією та обчислення значення зсуву. На цьому етапі використовується дуальна форма SVM (у рівнянні (8)), щоб сформулювати загальну оптимізацію за опуклою функцією, як показано в рівняннях (9) та (10).

Під час фази ініціалізації параметрів етапу 2 обчислюються кілька параметрів у рівняннях (9) та (10), включаючи параметри функції об'єкта (тобто, N, u), параметри обмежень (тобто, G, h, D, z_{const}), та інші параметри (α, Gr, допустима точка). На етапі оптимізації за опуклою функцією значення α та Gr обчислюються для кожної ітерації за допомогою методу розкладання послідовної мінімальної оптимізації (SMO). Після досягнення максимальної кількості ітерацій обчислюється значення зсуву z на останній фазі обчислення остаточного значення зсуву. Це значення зсуву використовується для визначення перетину гіперплощини.

Для етапу 2 було розроблено універсальний оптимізатор опуклої функції, використовуючи ту саму конвенцію позначення параметрів, що й загальна форма опуклої функції з рівняння (10). У цьому випадку позначення z та z_{const} відповідають значенням зсуву (у рівнянні (2)) та значенням обмежень (у рівнянні (10)) відповідно. Вищезгадані параметри об'єктивної функції та обмежень обчислюються на етапі 2 (фаза 1).

Параметри об'єктивної функції: Параметр N (у рівняннях (9) та (10)) - це матриця розміром mAm, яка обчислюється за допомогою скалярного добутку вихідних міток $B_x.B_y$. Змінна B - це вихідна мітка, отримана з набору даних, і вона відображає клас вихідних даних, як у рівнянні (1). На етапі 1, значення B зазвичай заздалегідь завантажується до BRAM разом із вхідними зразками. На етапі 2, результат скалярного добутку $(B_x.B_y)$ множиться на результат матриці ядра $K(A_x, A_y)$, щоб отримати матрицю N (у рівнянні (18)). У цьому випадку, як показано на рисунку 3.8, модулі для обчислення матриці N складаються з одиниці MAC та множника (тобто одиниця MAC1 та Mul1, на рисунку 3.8). Елементи матриці N зберігаються в BRAM та доступні за допомогою двох окремих генераторів адрес, щоб створити два набори матриць N_{set1} та N_{set2}. Для аналізу великих даних розмір матриці N збільшується у квадратичній залежності від зростання кількості вхідних даних; отже, виконання опуклої оптимізації за допомогою звичайних методів, таких як методи внутрішньої точки, стає обчислювально складним завданням. Щоб подолати цю проблему, використовується певний метод розкладу, щоб виконати опуклу оптимізацію [39]. З цим методом розкладу, на будь-якому етапі оптимізаційного процесу, вибираються два робочих набори.

Цей метод розкладу деталізується у наступній фазі опуклої оптимізації. Параметр u (у рівняннях (9) та (10)) є матрицею розміром mA1. Матриця N є масивом одиниць, як у рівнянні (19). З метою простоти під час проектування кожний елемент матриці N залишається сталою, рівною 1.

$$N_m = K \begin{bmatrix} B_0\,B_0 & B_0\,B_1 & \dots & B_0\,B_m \\ B_1\,B_0 & B_1\,B_1 & \dots & B_1\,B_m \\ \vdots & \vdots & \ddots & \vdots \\ B_m\,B_0 & B_m\,B_1 & \dots & B_m\,B_m \end{bmatrix} \tag{18}$$

$$u_{m,1} = (1, 1, \dots, 1) \times Q \tag{19}$$

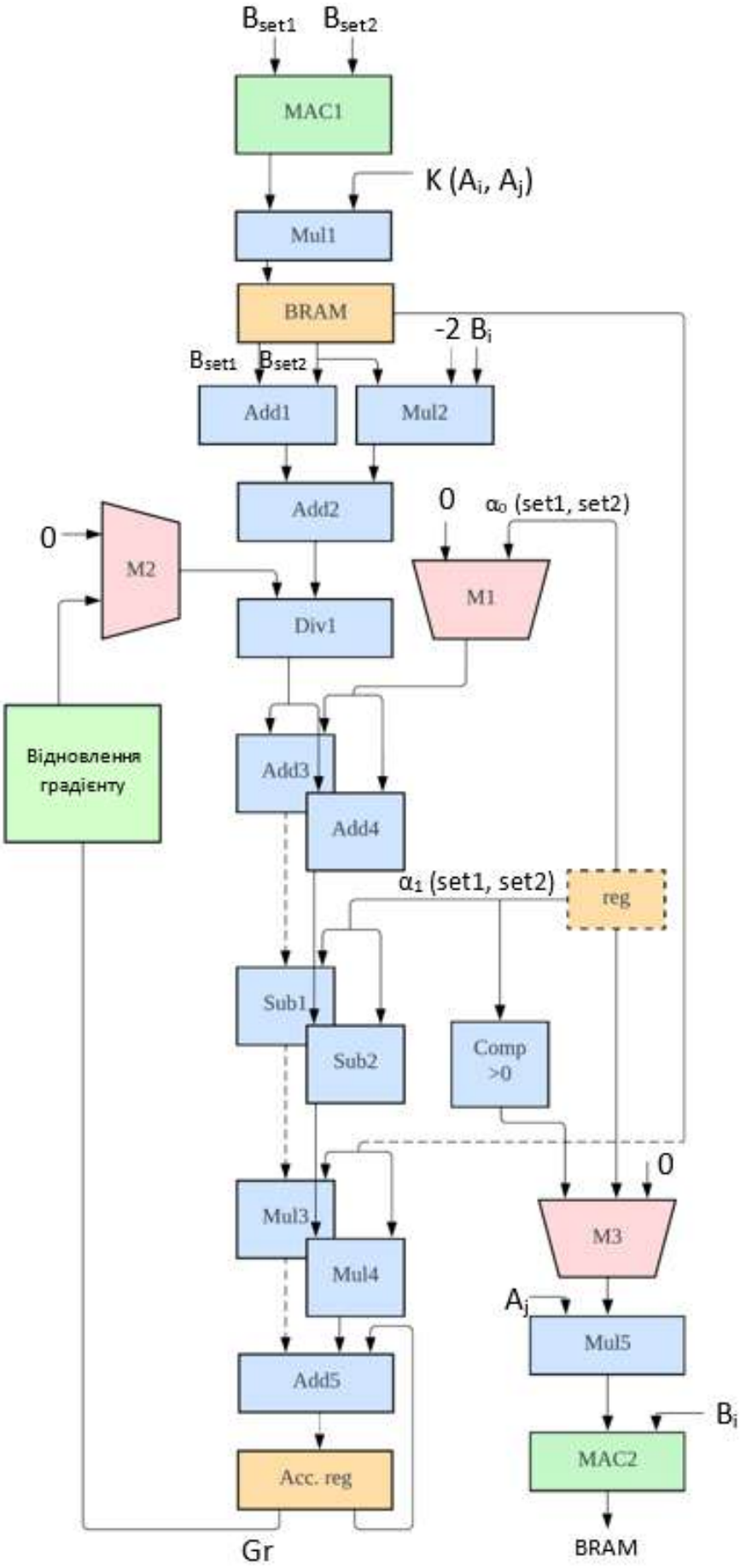


Рисунок 3.8 – Шлях даних для Опуклої Оптимізації

Параметри обмежень. У рівнянні (10), об'єктивна функція піддається області обмеження (яке складається з параметрів G та h) та рівності обмеження (яке складається з D та z_{const}). У цьому випадку, параметр D є матрицею розміром 1Am у рівнянні (10), яка збігається з вихідною міткою y у рівнянні (9). Крім того, параметр z_{const} у рівнянні (10) встановлено на нуль у рівнянні (9), а параметр G у рівнянні (10) є постійним значенням 1. Максимальний поріг області обмеження - параметр h у рівнянні (10) (тобто параметр W у рівнянні (9)). Значення W впливає на загальну швидкість та ефективність алгоритму SVM на основі опуклої оптимізації.

Інші параметри. Параметр α у рівнянні (9), який відповідає A у рівнянні (10), важливий для визначення опорних векторів у алгоритмі SVM на основі опуклої оптимізації. Оскільки координатні розміри опорних векторів визначають орієнтацію гіперплощини, значення α для всіх вхідних зразків ініціалізуються нулями перед проведенням опуклої оптимізації. Метою проведення опуклої оптимізації є обчислення мінімального значення для α для кожного вхідного зразка. На основі вищезазначеного порогового значення (W) вхідні зразки зі значенням α, що перевищує порогове значення, будуть вважатися опорними векторами.

Оскільки розмір матриці N зростає експоненційно зі збільшенням кількості вхідних зразків, у кожній ітерації обчислюються α за допомогою двох робочих наборів (або вхідних зразків). У цьому випадку для початкової точки вибирається допустимий набір (0, 0), який є першим вхідним зразком набору даних; а значення Gr (градієнта) ініціалізується як нуль. Кут нахилу градієнта до мінімального значення зазвичай відповідає напрямку градієнтного спуску; тому значення градієнта використовується для вибору наступного робочого набору.

Після фази ініціалізації параметрів виконується фаза оптимізації. Під час фази оптимізації проводяться п'ять операцій. Схема обробки даних для оптимізації показана на рисунку 3.8, яка складається з декількох суматорів, множників, віднімачів, модулів MAC, дільників, регістрів накопичувачів, компараторів, та мультиплексорів. Процес оптимізації (як в рівнянні (9)) полягає в пошуку

мінімального значення α. У цьому випадку значення α обчислюється за допомогою модулів Add1 до Add4, Mul1, Div1 та M1 (мультиплексор), як показано на рисунку 3.8. Для знаходження мінімального значення α використовуються додаткові компаратори, мультиплексори та модулі відновлення градієнта.

Спочатку, під час першої ітерації для пошуку мінімального значення α, два набори вхідних даних, N_{set1} та N_{set2} (збережені в BRAM), одночасно доступні за допомогою двох окремих генераторів адрес. Далі, результат операції додавання (з використанням Add1 на рисунку 3.8) N_{set1} та N_{set2} додається (з використанням Add2) до результату множення (тобто Mul2).

Потім значення градієнта ділиться (за допомогою Div1) на результат другої операції додавання (з використанням Add2). Оскільки значення Gr (градієнт) ініціалізоване нулем, початковий результат поділу також дорівнює нулю. Обчислення параметра градієнта модифіковане за допомогою модуля відновлення градієнта на рисунку 3.8.

Внутрішня архітектура модуля відновлення градієнта показана на рисунку 3.9. Під час першої ітерації значення α дорівнює нулю. Вихідна (або результат) операції ділення (за допомогою Div1 на рисунку 3.8) розглядається як значення зсуву, яке в свою чергу використовується для оновлення значення α. Окрім першої ітерації, де початкове значення α дорівнює нулю, починаючи з другої ітерації, вказане значення зсуву додається до значення α, обчисленого на попередній ітерації.

Наприклад, значення зсуву, отримане з другої ітерації, додається до значення α, згенерованого з першої ітерації, і так далі. У цьому випадку дві операції додавання виконуються паралельно (з використанням Add3 та Add4), щоб згенерувати нові значення α для кожного робочого набору.

Нові значення α зберігаються в тимчасовому регістрі (тобто reg на рисунку 3.8), щоб використовувати їх для майбутніх ітерацій. Далі, паралельно виконуються дві операції віднімання (за допомогою Sub1 та Sub2), щоб знайти різницю між новим та попереднім значенням α, яке надає відхилення між послідовними ітераціями. Потім нові значення α множаться паралельно (за

допомогою Mul3 та Mul4) на N_{set1} та N_{set2}, щоб знайти значення градієнта (Gr). Ці значення Gr зберігаються відповідно у внутрішньому ОЗП для подальших ітерацій та операцій

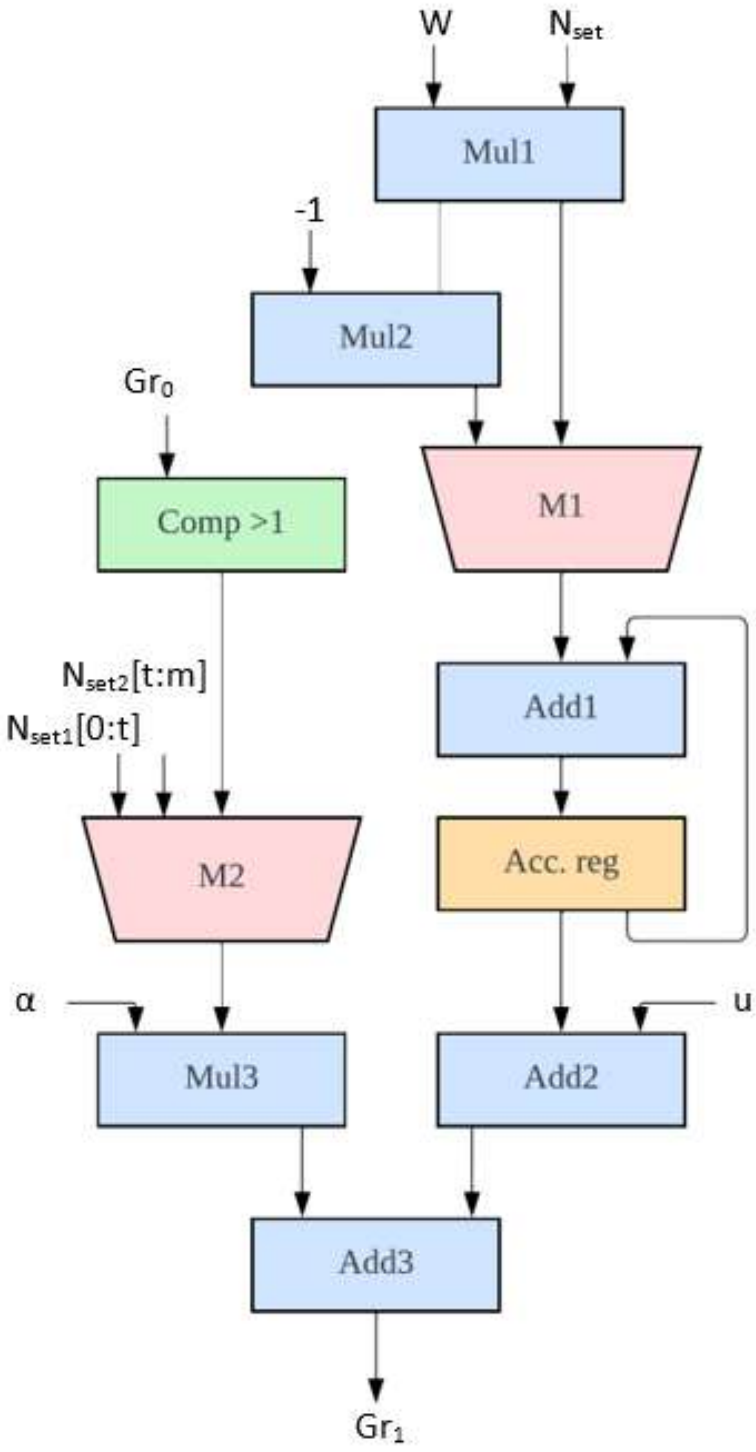


Рисунок 3.9 – Внутрішня архітектура обчислення відновленого градієнта

Знаходження мінімального значення для цільової функції в рівнянні (9) - це ітеративний процес. Цей процес триває до досягнення максимальної кількості ітерацій, яку, як правило, визначає користувач. Після досягнення максимальної кількості ітерацій значення α порівнюються з визначеним користувачем пороговим значенням (W) 10-3 за допомогою модуля Comp (на рисунку 3.8). У цьому випадку значення α, які перевищують порогове значення, вважаються векторами підтримки, а значення α, які менші за порогове значення, відкидаються. Ці вектори підтримки важливі для визначення орієнтації гіперплощини. Після отримання оптимального розв'язку для α (тобто мінімального значення α з рівняння (9)), це оптимальне значення α пересилається через мультиплексор (M3) до множника (Mul5), за яким слідує модуль MAC (MAC2) для обчислення вагових векторів (v) в рівнянні (7). Ці вагові вектори та значення α також зберігаються в ОЗП, а також в DDR3-SDRAM для наступного етапу тестування.

Внутрішня архітектура (або шлях даних) обчислення відновленого градієнта показана на рисунку 3.9. У випадку, якщо оптимізація не збігається до мінімального значення α під час ітеративного процесу опуклої оптимізації використовується модуль відновлення градієнта (на Рисунку 3.8) для коригування (або оновлення) значення параметра градієнта (Gr) і повторення процесу опуклої оптимізації. Коригування значень параметра градієнта залежить від значень α. Як показано на рисунку 3.9, модуль компаратора перевіряє, чи є напрямок спуску градієнта поточного робочого набору позитивним або від'ємним. У випадку з N_{set1}, якщо напрямок спуску градієнта позитивний, тоді решта значень N_{set1} (від t до m) призначаються від поточної кількості активних наборів до залишку кількості вибірки даних; і якщо напрямок спуску градієнта від'ємний, тоді робочий набір скидається для вибору із спочатку заданого набору (від 0 до t). Далі, отриманий N_{set1} вибирається через мультиплексор (M2), який множиться (за допомогою Mu13) на значення α. Потім виконується операція додавання (з використанням Add3) для отриманих результатів від Add2 та результатів від Mu13, щоб отримати оновлене значення Gr. Це оновлене значення Gr зберігається в BRAM для подальших ітерацій та аналізу, тобто сигнал "Gr", показаний на Рисунку 3.8.

Останній крок етапу 2 - це фаза обчислення значення зсуву. Значення зсуву z в рівнянні (2) також відоме як значення зміщення, яке представляє перетин гіперплощини з відліку. Значення зсуву обчислюється за наступним рівнянням (20):

$$z = -\frac{1}{2}\left[\max_{\{i|B_y=-1\}}\left(\sum_{y=1}^{m}\alpha_x BK(A_x, A_y)\right) + \min_{\{x|B_y=+1\}}\left(\sum_{y=1}^{m}\left(\alpha_x B_y K(A_x, A_y)\right)\right)\right] \quad (20)$$

$$\mathrm{z} = \frac{\left(\sum_{x=1}^{m} B_x - \sum_{x=1}^{m}\alpha_x B_{sv} K(A_x, A_y)\right)}{\text{кількість опорних векторів}, n_{sv}} \quad (21)$$

Рівняння (20) модифікується для включення суми всіх опорних векторів, а потім ділиться на загальну кількість опорних векторів для отримання середнього значення, як у рівнянні (21).

Як показано, рівняння (21) надає середні значення для всіх опорних векторів, тоді як рівняння (20) ідентифікує максимальне та мінімальне значення для кожного класу опорних векторів.

Внутрішня архітектура (або шлях даних) обчислення зсуву представлена на рисунку 3.10, яка включає модуль MAC, суматор, множник, віднімання, дільник та накопичуючий регістр з зворотнім зв'язком до додавання. Як показано на рисунку 3.10, результати (елементи) матриці ядра ($K(A_i, A_j)$, отримані на етапі 1) множаться на значення α (отримане на етапі 2).

Результат модуля Mul множиться на значення Bsv (яке є вихідною міткою в рівнянні (1), що відповідає опорному вектору) і сумується за допомогою модуля MAC. Шлях даних Mul та MAC відповідає другому доданку в рівнянні (21).

Мітка виходу опорного вектора (Bsv) подається через модуль додавання та накопичуючий регістр (як показано на рисунку 3.10), для отримання першого доданка в рівнянні (21). Результат модуля MAC віднімається від результату накопичувача, щоб отримати чисельник у рівнянні (21).

Після того, результат віднімача ділиться на загальну кількість опорних векторів (nsv), щоб отримати значення зсуву. Остаточне значення зсуву z

зберігається в енергоефективній пам'яті на криптосистемі BRAM, а також в пам'яті DDR3-SDRAM для подальшого аналізу.

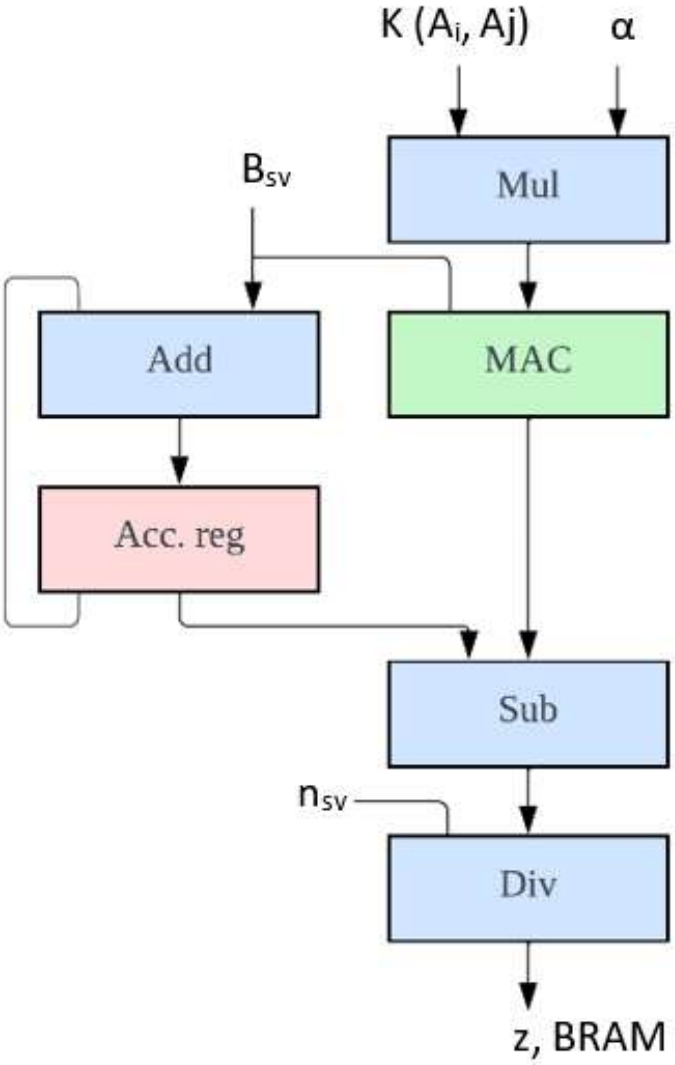


Рисунок 3.10 – Внутрішня архітектура обчислення зсуву

3.6.3 Етап 3: Тестування

Фаза 3, яка є останньою фазою, - це процес тестування. Зазвичай для класифікації вхідний набір даних розділяється на два зразки: навчальний і тестовий. Під час етапів 1 і 2 навчання виконується за допомогою навчального набору, тоді як на етапі 3 тестування виконується за допомогою тестового набору. Для процесу тестування вхідні вектори даних класифікуються у клас -1 (мінус

один) або +1 (плюс один) на основі значення знака функції f (A) в рівнянні (3). У цьому випадку, якщо вихід рівняння (3) менше нуля (f (A) <0), тоді вхідні вектори тесту призначаються класу -1; і якщо вихід рівняння (3) більше нуля (f (A)> 0), тоді вхідні вектори тесту призначаються класу +1. Використовується наступна формула, яка відома як рішення (рівняння (22)), для розробки стадії тестування (або класифікації) алгоритму SVM на основі опуклої оптимізації.

$$\emptyset(A_t) = sgn\{\sum_{x=1}^{m} \propto_x B_x K(A_x, A_t) + z\} \quad (22)$$

Шлях даних процесу тестування, який виконує остаточну класифікацію SVM, показано на рисунку 3.11. Як вже зазначалося, цей шлях даних розроблено на основі рівняння (22).

На цьому етапі, як і у рівнянні (22), необхідно виконати перевірку знака для тестування (класифікації) процесу. Як показано на рисунку 3.11, шлях даних для тестування (класифікації) включає множник, модуль MAC, суматор, мультиплексор та компаратор. Спочатку вектори тесту (з тестового зразка) передбачаються в BRAM з DDR3-SDRAM та пересилаються до блока ядра у формі конвеєра. Як детально описано на етапі 1, модуль ядра відображає ці вектори тесту в простір ознак. Дві згадані вище кроки виконуються на етапі 1.

На етапі 3 повторно використовуються модулі передбачення та ядра з етапу 1 для зменшення загальної площі, зайнятої апаратним забезпеченням.

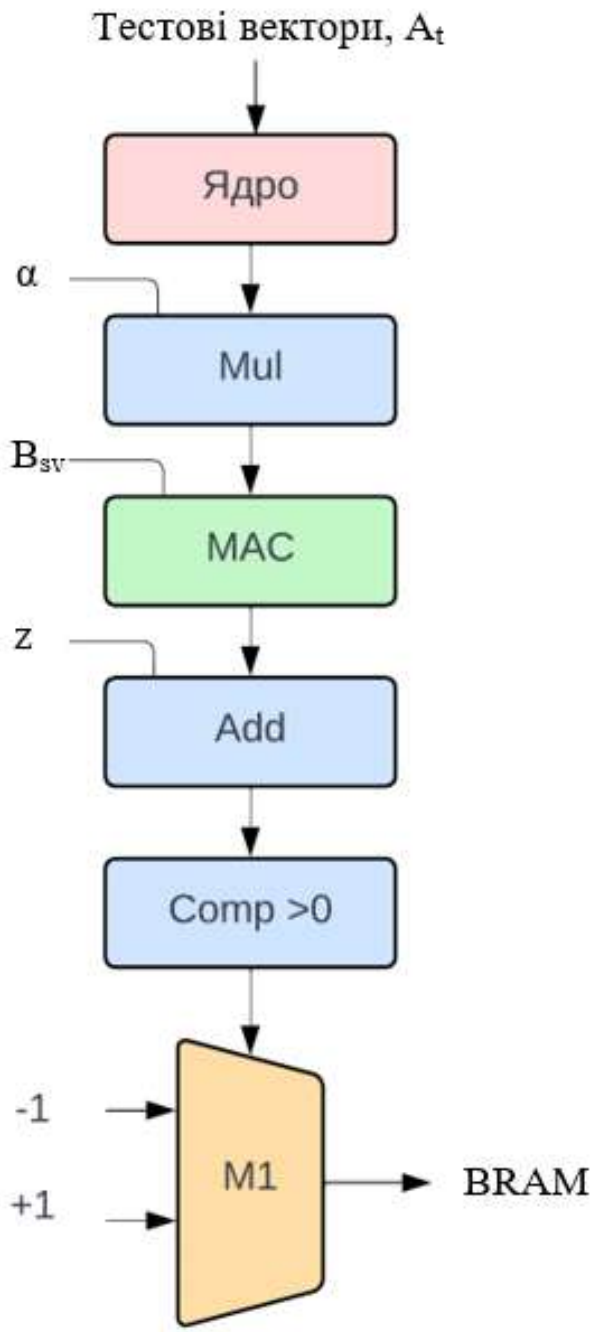


Рисунок 3.11 – Шлях даних для процесу випробування

На етапі 3, як показано на Рисунку 3.11, результат обчислення ядра, який є матрицею ядра, спочатку множиться на значення α (отримане на етапі 2) за допомогою модуля Mul. Потім результат модуля Mul пересилається в модуль MAC для множення на вихідну мітку підтримуючого вектора (Bsv), а потім виконується операція додавання, що відповідає Рівнянню (22). Після того, значення зсуву (z)

(отримане на етапі 2 (у Рівнянні (21)) додається (за допомогою модуля Додавання) до кінцевого результату сумування модуля MAC. Потім результат додавання пересилається до компаратора для визначення того, чи результат додавання більше або дорівнює нулю. На основі результатів компаратора вхідні вибірки тестових даних призначаються до класу +1 або -1 за допомогою мультиплексора.

3.7 Висновки

Досліджено та проаналізовано основи створення SVM-класифікатора для аналізу даних на базі FPGA.

Виявлено, що використання методу опорних векторів у поєднанні з опуклою оптимізацією дозволяє ефективно розв'язувати задачі класифікації навіть для складних та нелінійних наборів даних.

Архітектура на рівні системи була детально проаналізована з метою розробки оптимізованої системи аналізу даних на базі FPGA.

Встановлено, що ефективне поєднання апаратних та програмних компонентів дозволяє досягти значного прискорення процесу аналізу даних та забезпечує високу швидкодію системи.

Для реалізації запропонованого методу було створено вбудований апаратний дизайн з урахуванням специфіки завдань аналізу даних, що дозволило створити ефективний інструмент для реалізації SVM-класифікатора на основі опуклої оптимізації на FPGA.

Виявлено, що використання апаратних ресурсів дозволяє значно прискорити процес аналізу даних та покращити загальну швидкодію системи.

4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ЗАПРОПОНОВАНОГО МЕТОДУ

Було проведено тестування для оцінки можливості та ефективності запропонованого вбудованого апаратного та програмного забезпечення для алгоритму машинного навчання SVM на основі опуклої оптимізації (СО), з точки зору швидкодії (прискорення), точності, а також масштабованості для роботи з різними наборами даних різних розмірів. Було виміряно точність класифікації та швидкодію, використовуючи наступні рівняння (23) та (24) відповідно.

Метрика масштабованості призначена для демонстрації можливості запропонованого вбудованого апаратного та програмного забезпечення працювати з різними наборами даних різного розміру, різною кількістю атрибутів та іншими змінними параметрами, які часто зустрічаються у багатьох наборах даних для застосувань машинного навчання.

:

$$\text{Точність (в \%)}, (B_x, \hat{B}) = \frac{1}{n_x}\sum_{x=0}^{n_x-1} 1(B_x == \hat{B}) \times 100 \quad (23)$$

$$\text{Прискорення } = \frac{\text{Програмний час виконання}}{\text{Апаратний час виконання}} \quad (24)$$

Під час початкової фази проектування було порівняно отримані результати з результатами відкритого вихідного коду Python [36], щоб перевірити правильність, функціональність та точність запропонованого вбудованого апаратного та програмного забезпечення.

Час виконання отримується у тактових циклах та перетворюється на секунди. Запропоновані архітектури вбудованого обладнання (як апаратне, так і програмне забезпечення) виконуються на Altera Cyclone V FPGA, а запропоновані архітектури вбудованого програмного забезпечення виконуються на вбудованому процесорі ARM Cortex-A9 на тій же FPGA, тоді як код Python виконується на настільному комп'ютері з процесором Intel-i7, що працює на частоті 2,3 ГГц.

У даному дослідженні тестування було проведено для оцінки запропонованого вбудованого апаратного та програмного забезпечення на двох різних наборах даних для тестування: набір даних «Prediction of Pulsar Star» [40] та набір даних «Iris Flowers» [39] для застосувань машинного навчання.

Ці набори даних зберігаються у DDR3-SDRAM та форматуються відповідно для відмежування між вхідними ознаками та вихідними мітками. Розмір даних вимірюється з урахуванням кількості векторів/вибірок (n) та кількості вимірів/ознак (m) у кожному векторі.

У даному дослідженні розміри даних змінюються для вивчення їх впливу на точність, прискорення та масштабованість.

Для всіх експериментів набори даних було розділено на два набори: навчальний та тестовий. Тестовий набір розглядається як відсоток набору даних для вивчення точності класифікації.

У цьому випадку навчальний набір змінюється від 10% до 90% з кроком 10%. Експериментальні результати щодо часів виконання, прискорення та точності класифікації для запропонованого вбудованого апаратного та програмного забезпечення представлені в таблицях 4.1–4.6.

Окрім зміни розмірів даних для тестування та навчання, також змінюється кількість ітерацій (для пошуку мінімуму) під час навчального процесу, щоб дослідити здатність та прискорення процесу опуклої оптимізації для знаходження цих значень мінімумів.

4.1 Аналіз використання ресурсів

Для оцінки можливості та ефективності використання площі запропонованих вбудованих апаратних архітектур було проведено аналіз вартості простору (використання ресурсів).

У цьому випадку після процесу реалізації було отримано значущі параметри використання ресурсів, включаючи кількість зайнятих сегментів, кількість BRAMs, тоді як кількість зайнятих сегментів зазвичай складається з регістрів сегментів та

LUTs. Ці статистичні дані використання ресурсів для запропонованого вбудованого апаратного забезпечення представлені в Таблиці 4.1. Як показано, загальна кількість зайнятих сегментів, кількість DSP48E1 та кількість BRAMs становлять відповідно 5216, 110 та 118. Беручи до уваги загальну кількість логічних сегментів (37 680 сегментів) у Altera Cyclone V FPGA, запропонований апаратний дизайн займає лише 12.7% площі чіпа.

Під час початкової фази даного дослідження було розглянуто можливість та компроміс щодо використання регістрів порівняно з BRAMs для зберігання проміжних результатів.

Було виявлено, що використання BRAMs призводить до значного зменшення загальної кількості зайнятих сегментів на чипі порівняно з кількістю регістрів. Крім того, BRAMs необхідні для зберігання проміжних мінімальних значень під час послідовної мінімальної оптимізації (SMO). Для певних операцій (таких як векторні обчислення) було використано сегменти DSP48E1 для обчислень з одиночною точністю плаваючої коми. Це рішення також призводить до більш ефективного використання площі та меншої затримки тактування для операцій з плаваючою комою порівняно з використанням варіантів, що базуються виключно на логіці.

Використання вбудованих BRAMs на чипі (і у деяких випадках реєстрів) для зберігання проміжних результатів значно зменшує час виконання численних обчислень матриці, що властиві алгоритму машинного навчання SVM на основі CO, тим самим підвищуючи загальну швидкодію цього алгоритму.

Введені методи попереднього завантаження для зменшення затримки доступу до пам'яті та використання вбудованих BRAM для зберігання даних/результатів дійсно додають більше простору (тобто додаткових ресурсів).

Таким чином, важливо враховувати компроміс між швидкодією та обсягом пам'яті при проектуванні певних алгоритмів/методів, таких як алгоритм машинного навчання SVM на основі CO, для застосувань машинного навчання, які зазвичай вимагають обробки великого обсягу даних, особливо на вбудованих платформах з їх обмеженнями щодо площі.

4.2 Аналіз точності класифікації

Було проведено тестування для оцінки точності класифікації запропонованих вбудованих конструкцій для алгоритму машинного навчання SVM на основі CO. У цьому випадку точність класифікації для алгоритму машинного навчання SVM на основі CO отримується з різними розмірами даних для максимальної кількості ітерацій 1000. Точність класифікації вимірюється за допомогою Рівняння (23).

Для вимірювання точності класифікації набори даних було розділено на два набори: навчальний та тестовий. Навчальний набір змінюється від 10% до 90% з кроком 10%, щоб дослідити точність класифікації. Крім того, для лінійної, поліноміальної та гаусової радіальної базисної функції (RBF) на Етапі 1 було досліджено та обрано наступні специфікації: параметр штрафу (W) встановлено на 1; ступінь полінома (d) встановлено на 2 з коефіцієнтом 1; та β встановлено на 0.0001.

Зміна цих параметрів може потенційно призвести до проблем недооцінювання та переоцінювання. Недооцінювання виникає, коли SVM узагальнює основні ознаки даних; тоді як переоцінювання виникає, коли SVM навчається бути чутливим до шуму [13].

В результаті для даного дослідження набори даних було розділено і використано метод перехресної перевірки, щоб вибрати відповідні константи.

Методи перехресної перевірки дозволяє навчити SVM шляхом розбиття набору даних, а також налаштувати вищезазначені параметри для отримання найкращих результатів точності. У цьому випадку, щоб уникнути проблем недооцінювання та переоцінювання, SVM навчається та тестується з різними параметрами (W, ступінь, гамма, кількість ітерацій), щоб побудувати кращий класифікатор даних.

Таблиці 4.1–4.6 містять результати точності класифікації (для вбудованих апаратних та програмних конструкцій) для загального алгоритму машинного навчання SVM на основі CO з використанням наборів даних «Pulsar» та «Iris

Flowers».

Результати точності подано у відсотках і представлено окремо для трьох різних математичних ядер на Етапі 1: лінійного (в Таблицях 4.1 і 4.4), поліноміального (в Таблицях 4.2 і 4.5) та гаусового RBF (в Таблицях 4.3 і 4.6). Результати точності представлені у сьомому стовпчику цих таблиць.

Таблиця 4.1. Вбудоване апаратне та програмне забезпечення: час виконання, прискорення, точність з використанням набору даних-експериментів «Pulsar» для Лінійного Ядра, з W = 1, s= 2, β = 0.0001.…

Розмір даних	Тестувальний набір (%)	Кількість векторів	Час виконання мікропроцесора	Апаратний час виконання	Прискорення	Точність (%)
8074	10	57	158692325	6895652	24,23	83,57
16148	20	114	352569874	13562325	24,02	92,2
24222	30	171	687562521	24562202	25,89	90,64
32296	40	228	687521232	32023256	40,99	91,78
40370	50	285	2114521452	35744523	58,62	92,51
48444	60	342	3256125322	43236586	72,2	92,01
56775	70	399	4215212321	61256965	68,25	91,99
64592	80	456	5884521451	84956256	68,55	90,88
71326	90	513	7654125000	101235629	75,2	93,01

Таблиця 4.2. Вбудоване апаратне та вбудоване програмне забезпечення: час виконання, прискорення, точність з використанням набору даних-експериментів «Pulsar» для Поліноміального Ядра, з W = 1, s= 2, β = 0.0001.

Розмір даних	Тестувальний набір (%)	Кількість векторів	Час виконання мікропроцесора	Апаратний час виконання	Прискорення	Точність (%)
8074	10	57	385695621	122562325	2,85	82,99

Кінець таблиці 4.2

16148	20	114	1874562325	134568956	12,54	87,85
24222	30	171	4654987123	155256585	28,25	92,85
32296	40	228	7987654123	184526523	41,22	87,99
40370	50	285	12456251211	222565258	56,52	85,86
48444	60	342	19562521458	268591325	72,52	92,78
56775	70	399	19854555852	323569585	61,2	92,15
64592	80	456	25062598532	385625102	66,52	92,15
71326	90	513	27896523658	460232650	61,2	97,54

Таблиця 4.3. Вбудоване апаратне та вбудоване програмне забезпечення: час виконання, прискорення, точність з використанням набору даних-експериментів «Pulsar» для Ядра Гаусса RBF, з W = 1, s= 2, β = 0.0001.

Розмір даних	Тестувальний набір (%)	Кількість векторів	Час виконання мікропроцесора	Апаратний час виконання	Прискорення	Точність (%)
8074	10	57	632562584	46523255	15,25	72,58
16148	20	114	1856589521	945200252	18,52	87,56
24222	30	171	2877458555	111252023	23,52	89,25
32296	40	228	4854555235	130252302	35,25	91,89
40370	50	285	7589568222	145202300	47,77	93,01
48444	60	342	10858200526	225203265	47,52	92,56
56775	70	399	15856958452	384252002	47,85	91,86
64592	80	456	22425125655	385236201	59,56	89,99
71326	90	513	25965232562	463256966	56,25	89,56

Таблиця 4.4. Вбудоване апаратне та вбудоване програмне забезпечення: час виконання, прискорення, точність з використанням набору даних-експериментів «Iris Flowers» для Лінійного Ядра, з W = 1, s= 2, β = 0.0001.

Розмір даних	Тестувальний набір (%)	Кількість векторів	Час виконання мікропроцесора	Апаратний час виконання	Прискорення	Точність (%)
289	10	36	339,293,000	10,589,052	33,2	64,89
579	20	71	1,002,549,999	30,265,650	34,1	68,95
868	30	106	2,088,900,000	52,608,548	38,99	69,56
1158	40	141	3,566,149,999	86,932,128	42,5	91,56
1448	50	176	5,243,630,000	102,803,892	51,2	78,2
1737	60	211	7,641,730,000	134,483,893	57,25	75,68
2027	70	246	10,094,000,000	162,238,668	63,52	97,2
2316	80	281	13,728,500,000	218,501,180	63,85	96,1
2356	90	316	19,554,900,000	311,316,697	61,25	99,9

Таблиця 4.5. Вбудоване апаратне та вбудоване програмне забезпечення: час виконання, прискорення, точність з використанням набору даних-експериментів «Iris Flowers» для Поліноміального Ядра, з W = 1, s= 2, β = 0.0001.

Розмір даних	Тестувальний набір (%)	Кількість векторів	Час виконання мікропроцесора	Апаратний час виконання	Прискорення	Точність (%)
289	10	36	454,131,374	127,739,374	4,2	64,23
579	20	71	1,109,957,922	124,685,905	8,5	68,52
868	30	106	2,412,208,381	132,320,757	16,59	68,25
1158	40	141	4,156,918,296	147,575,466	28,56	62,35
1448	50	176	6,088,204,069	163,878,344	35,68	75,2

Кінець таблиці 4.5

1737	60	211	8,680,224,863	182,982,124	48,52	75,6
2027	70	246	11,520,789,306	206,488,474	54,84	97,89
2316	80	281	15,680,125,496	239,989,386	66,2	99,9
2356	90	316	20,949,687,217	264,792,097	78,21	99,9

Таблиця 4.6. Вбудоване апаратне та вбудоване програмне забезпечення: час виконання, прискорення, точність з використанням набору даних-експериментів «Iris Flowers» для Ядра Гаусса RBF, з W = 1, s= 2, β = 0.0001.

Розмір даних	Тестувальний набір (%)	Кількість векторів	Час виконання мікропроцесора	Апаратний час виконання	Прискорення	Точність (%)
289	10	36	131,588,528	17,246,202	8,2	61,89
579	20	71	443,339,285	33,586,309	12,5	62,89
868	30	106	1,369,622,549	62,597,008	22,5	72,2
1158	40	141	2,732,821,550	102,237,992	26,85	74,89
1448	50	176	3,938,128,950	136,693,125	27,89	76,99
1737	60	211	8,127,266,954	241,523,535	34,2	67,1
2027	70	246	7,150,427,148	191,957,775	38,56	94,56
2316	80	281	20,918,761,361	528,652,043	40,01	97,89
2356	90	316	29,338,152,919	718,544,034	41,5	98,2

4.2.1 Аналіз точності класифікації при змінних розмірах даних

У результаті аналізу Таблиць 4.1, 4.2, 4.3, 4.4, 4.5 і 4.6 можна зазначити, що точність класифікації змінюється в залежності від різних наборів даних та різних відсотків навчальних наборів. Загалом, здається, що точність класифікації зростає зі збільшенням відсотка навчального набору для обох наборів даних. Наприклад,

точність класифікації збільшується: від 83–93% (у Таблиці 4.1) та від 66–100% (у Таблиці 4.4) з лінійним ядром; від 81–96% (у Таблиці 4.2) та від 66–100% (у Таблиці 4.5) з поліноміальним ядром; та від 74–90% (у Таблиці 4.3) та від 60–98% (у Таблиці 4.6) з гаусовим RBF ядром. З Таблиць 4.4, 4.5 і 4.6 видно, що набори даних Iris Flowers» мають точність класифікації 100%, коли відсоток навчального набору становить 90% від набору даних для лінійного та поліноміального ядр. З Таблиць 4.1, 4.2 і 4.3 видно, що набір даних-експериментів «Pulsar» досягає найкращої точності класифікації 96% з поліноміальним ядром, коли відсоток навчального набору становить 90% від набору даних. Варто відзначити, що результати точності класифікації однакові для запропонованих як вбудованих, так і апаратних програмних конструкцій.

Окрім запропонованих вбудованих апаратних та програмних архітектур, експерименти з точністю класифікації також були проведені на коді Python, що виконується на настільному комп'ютері. Графіки точності результатів дизайну коду Python при використанні лінійних, поліноміальних та гаусових RBF математичних ядер представлено на Рисунках 4.1 і 4.2 відповідно для наборів даних-експериментів «Pulsar» та «Iris Flowers» (з максимальною кількістю ітерацій 1000). Крім того, результати точності запропонованих конструкцій при використанні тих же математичних ядер (як представлено в Таблицях 4.1, 4.2, 4.3, 4.4, 4.5 і 4.6), також подані на Рисунках 4.1 і 4.2 для наборів даних-експериментів «Pulsar» та «Iris Flowers» відповідно.

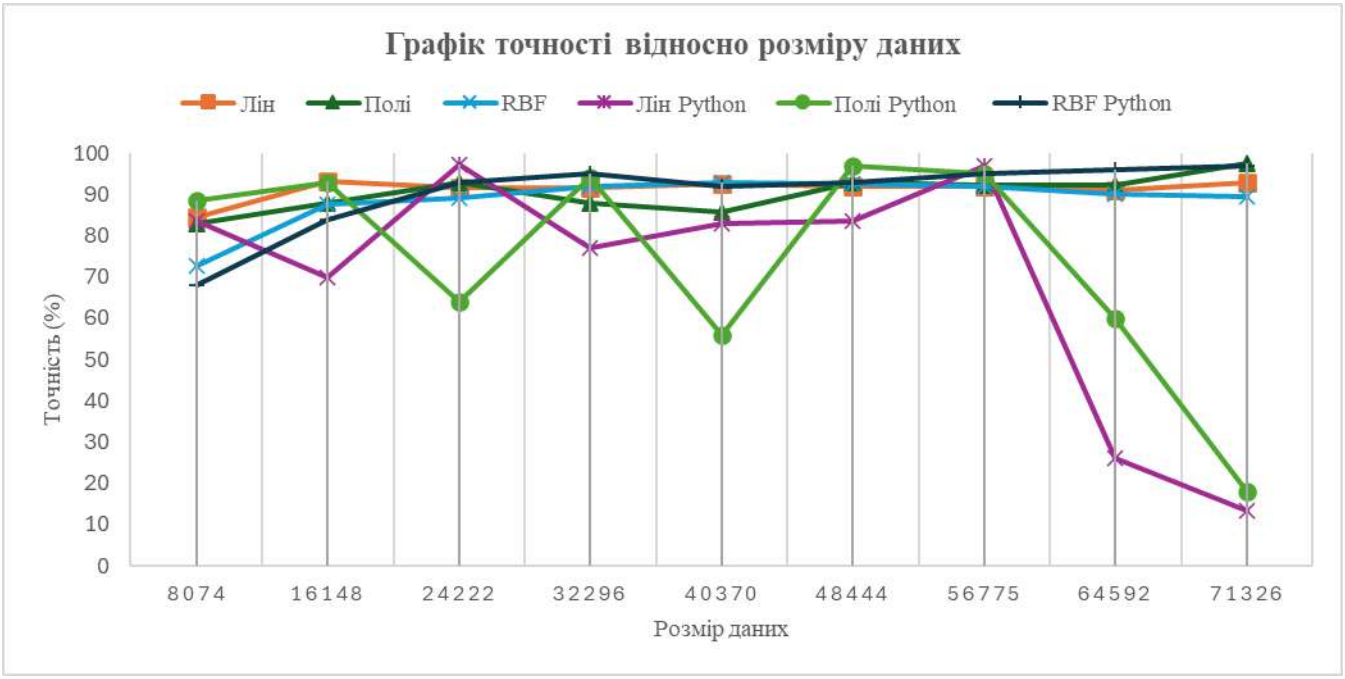


Рисунок 4.1 – Графік точності класифікації та розміру даних для набору даних-експериментів «Pulsar»

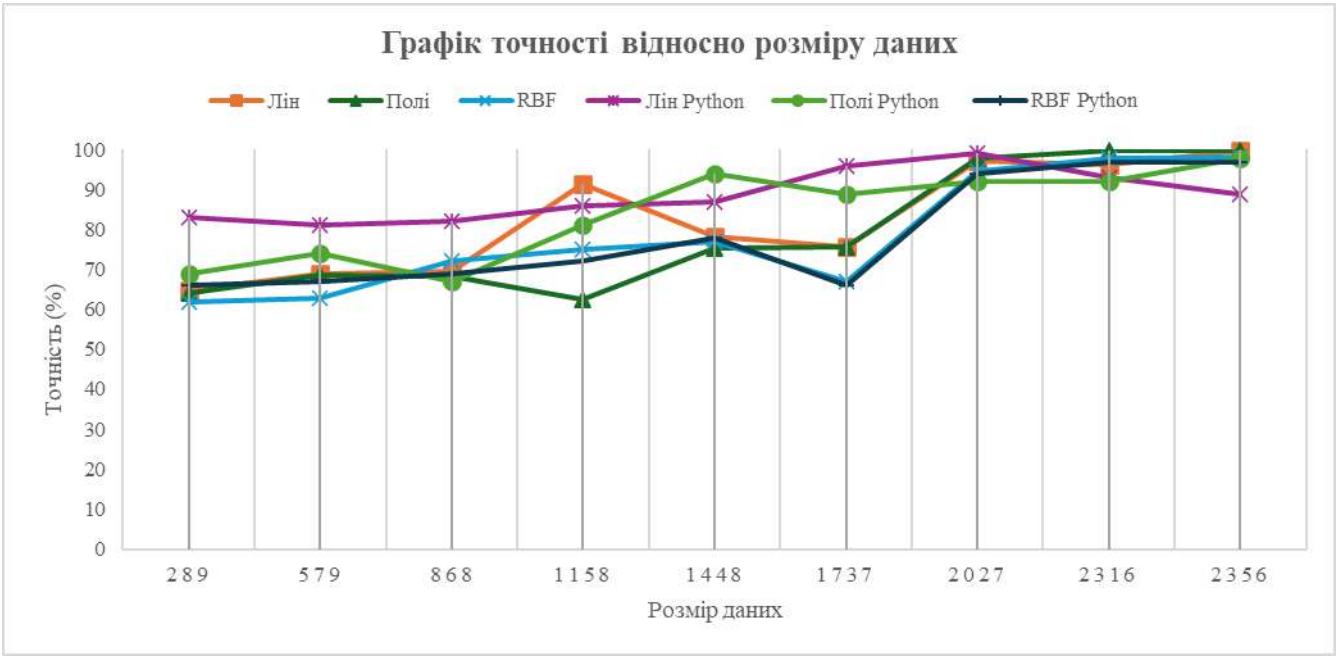


Рисунок 4.2 – Графік точності класифікації та розміру даних для набору даних-експериментів «Iris Flowers»

З Рисунка 4.1 видно, що точність класифікації кодів Python не збільшується зі зростанням відсотка навчального набору для набору даних-експериментів «Pulsar».

У цьому випадку результати точності класифікації коду Python здаються дуже непослідовними порівняно з результатами точності класифікації запропонованих вбудованих конструкцій. Крім того, результати точності класифікації здаються непослідовними особливо між двома кодами Python: один з лінійним ядром, а інший з поліноміальним ядром. Навпаки, результати точності класифікації для запропонованих вбудованих конструкцій практично накладаються для всіх трьох ядер.

Візуально, як показано на Рисунку 4.2, результати точності класифікації коду Python зростають зі збільшенням відсотка навчального набору для набору даних-експериментів «Iris Flowers». У цьому випадку результати точності класифікації здаються трохи непослідовними, особливо між двома кодами Python: один з лінійним ядром, а інший з поліноміальним ядром; тоді як результати точності класифікації для запропонованих вбудованих конструкцій практично накладаються для всіх трьох ядер. Як ілюстровано, результати точності класифікації коду Python з гаусовим RBF ядром досить схожі на запропоновані вбудовані конструкції для обох наборів даних-експериментів.

Непослідовності у коді Python (на Рисунку 4.1) можуть бути пов'язані з проблемами переоцінювання при виконанні з постійною кількістю ітерацій (тобто 1000 ітерацій). З проведених експериментів було виявлено, що ці коди Python збігаються до мінімального значення з вільною кількістю ітерацій. У цьому випадку кількість ітерацій досить велика, наприклад, кількість ітерацій становить 18 000 для 10% навчального набору даних, і 13 592 546 для 60% навчального набору даних.

4.2.2 Аналіз точності класифікації зі змінною кількістю ітерацій

Зазначені вище результати точності класифікації отримані з різними розмірами даних та постійною кількістю ітерацій. Було проведено додаткові експерименти, щоб проаналізувати результати точності класифікації зі змінною кількістю ітерацій та з постійним розміром даних. У цьому випадку Було обрано

розмір даних навчального набору 50% для обох наборів даних «Pulsar» та «Iris Flowers». В даному випадку було змінено максимальну кількість ітерацій від 100 до 2500 з кроком 500, щоб знайти мінімальне значення. Результати точності класифікації запропонованих конструкцій, а також кодів Python при використанні лінійних, поліноміальних та RBF математичних ядер показані на Рисунках 4.3 і 4.4 для наборів даних-експериментів «Pulsar» та «Iris Flowers» відповідно.

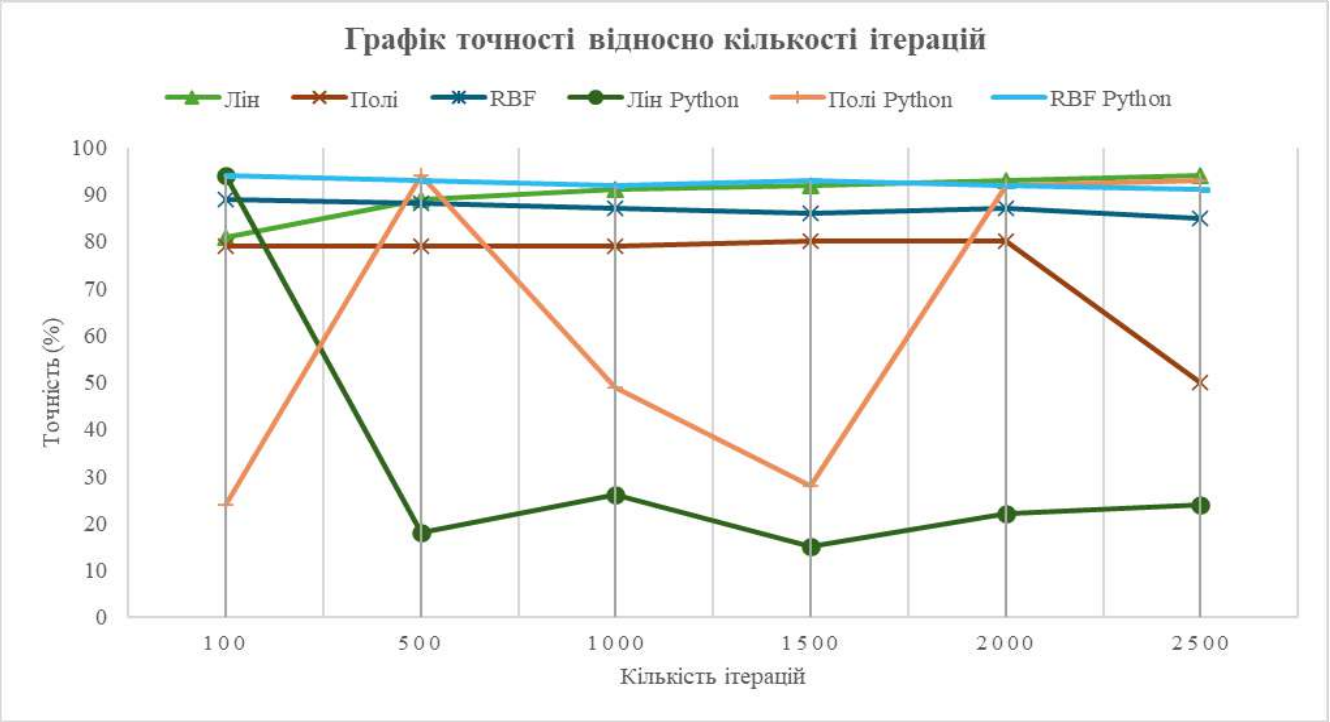


Рисунок 4.3 – Графік точності класифікації та кількості ітерацій для набору даних-експериментів «Pulsar»

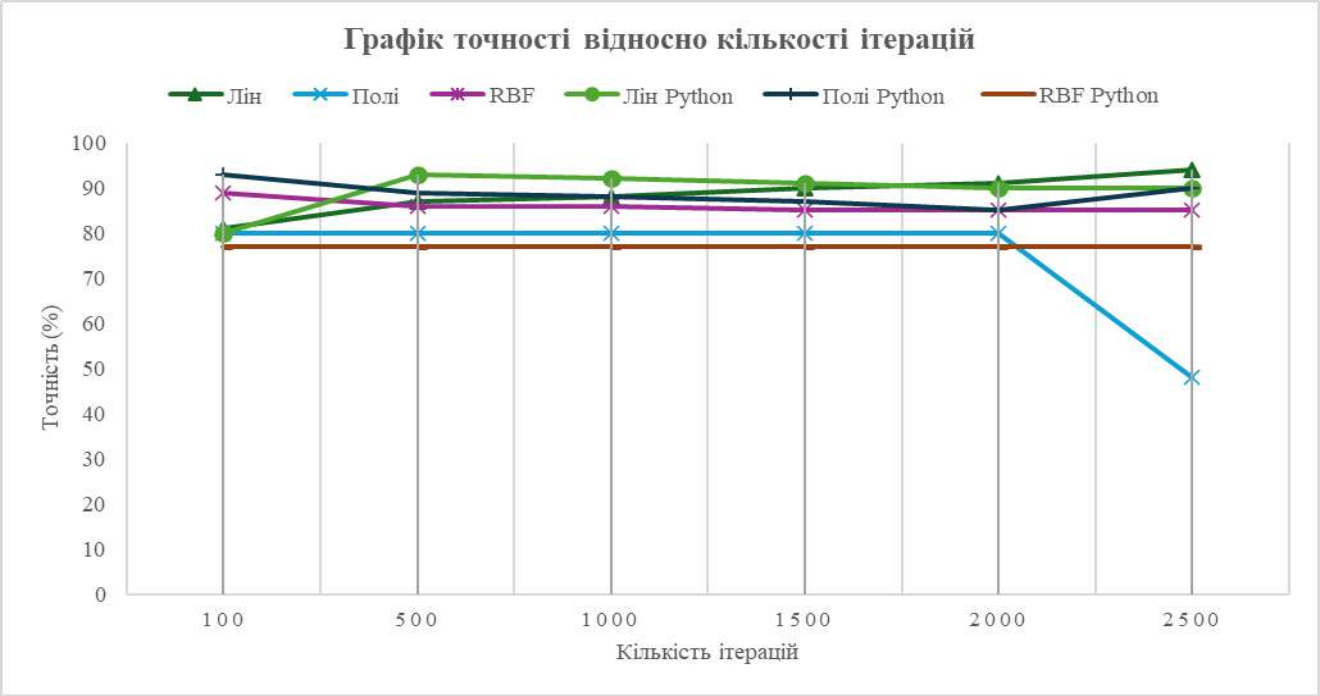


Рисунок 4.4 – Графік точності класифікації та кількості ітерацій для набору даних-експериментів «Iris Flowers»

Як показано на Рисунку 4.13, результати точності класифікації коду Python виявилися дуже непослідовними порівняно з результатами точності класифікації запропонованих вбудованих конструкцій при змінній кількості ітерацій для набору даних-експериментів «Pulsar». Крім того, результати точності класифікації здаються дуже непослідовними, особливо між двома кодами Python: один з лінійним ядром, а інший з поліноміальним ядром для набору даних-експериментів «Pulsar». Візуально, як показано на Рисунку 4.3, результати точності класифікації запропонованих вбудованих конструкцій майже однакові; наприклад, для обраного методу SVM на основі CO з лінійним ядром точність варіюється від 83% до 92%, при максимальній кількості ітерацій від 100 до 2500 відповідно, тоді як з RBF ядром Гауса точність залишається на одному рівні - 90%, незалежно від кількості ітерацій. Для запропонованого методу SVM на основі CO з поліноміальним ядром також точність залишається на одному рівні - 80%, при максимальній кількості ітерацій від 100 до 2000. Однак точність зменшується від 80% до 50%, при максимальній кількості ітерацій від 2000 до 2500, через проблему переоцінки. Як показано, результати точності класифікації з RBF ядром Гауса практично не зазнають

драматичних змін в залежності від змінної кількості ітерацій і практично однакові як для коду Python, так і для запропонованих вбудованих конструкцій.

Для наборів даних «Iris Flowers» спостерігається, що процес оптимізації збігається до мінімуму з меншою кількістю ітерацій, наприклад, за 100 ітерацій у деяких випадках. Внаслідок цього вплив кількості ітерацій на точність є незначним, як показано на Рисунку 4.4.

4.3 Аналіз часу виконання

Для оцінки швидкодії запропонованого вбудованого апаратного забезпечення було розроблено та реалізовано вбудоване програмне забезпечення для методу SVM на основі CO. Вбудоване програмне забезпечення виконується на м'якому процесорі ARM Cortex-A9 на тій же самій платформі розробки DE1 Soc. Час виконання як для вбудованого апаратного, так і для вбудованого програмного забезпечення отримується за допомогою таймера AXI, який працює на частоті 100 МГц на платі DE1 Soc.

Цей час виконання вимірюються в реальному часі, поки запропоновані вбудовані конструкції фактично працюють на чіпі. У цьому випадку Було розроблено таймер AXI в режимі каскаду для вимірювання точного часу виконання всіх трьох етапів алгоритму SVM на основі CO.

Основною причиною цього є те, що в деяких сценаріях, особливо для великих наборів даних, час виконання перевищує допустиме значення лічильника таймера AXI. Для вирішення проблеми переповнення лічильника таймера AXI розробляється з використанням двох таймерів у режимі каскаду.

Час виконання для запропонованих вбудованих апаратних та програмних конструкцій отримано з різними розмірами даних для максимальної кількості ітерацій 1000.

Час виконання для загального алгоритму SVM на основі CO з використанням наборів даних-експериментів «Pulsar» та «Iris Flowers» представлено у відповідних Таблицях 4.1, 4.2, 4.3, 4.4, 4.5 та 4.6.

Аналогічно до результатів точності класифікації, отримано три набори часу виконання для вбудованих програмних та апаратних конструкцій окремо, коли використовується три різні математичні ядра для Етапу 1, тобто лінійне (у Таблицях 4.1 та 4.4), поліноміальне (у Таблицях 4.2 та 4.5) та RBF Гауса (у Таблицях 4.3 та 4.6).

Час виконання для кожного набору (як для вбудованих апаратних, так і для програмних) вимірюється 10 разів, і середнє значення представлено у відповідних стовпцях 4 та 5 цих таблиць.

Час виконання для вбудованих апаратних та програмних конструкцій алгоритму SVM на основі CO з використанням лінійних, поліноміальних та гаусових RBF ядер представлені на Рисунках 4.5 і 4.6 відповідно для наборів даних-експериментів «Pulsar».

Як показано, час виконання майже експоненційно збільшується зі зростанням розмірів даних як для вбудованих апаратних, так і для програмних конструкцій. Дещо схожі результати отримуються при використанні набору даних «Iris Flowers». SVM на основі CO з лінійним ядром займає менше часу виконання порівняно з поліноміальним ядром.

Як показано на Рисунках 4.5 і 4.6, час виконання найбільший для методу SVM з поліноміальними ядрами, тоді як час виконання найменший для методу SVM з лінійними ядрами для набору даних-експериментів «Pulsar».

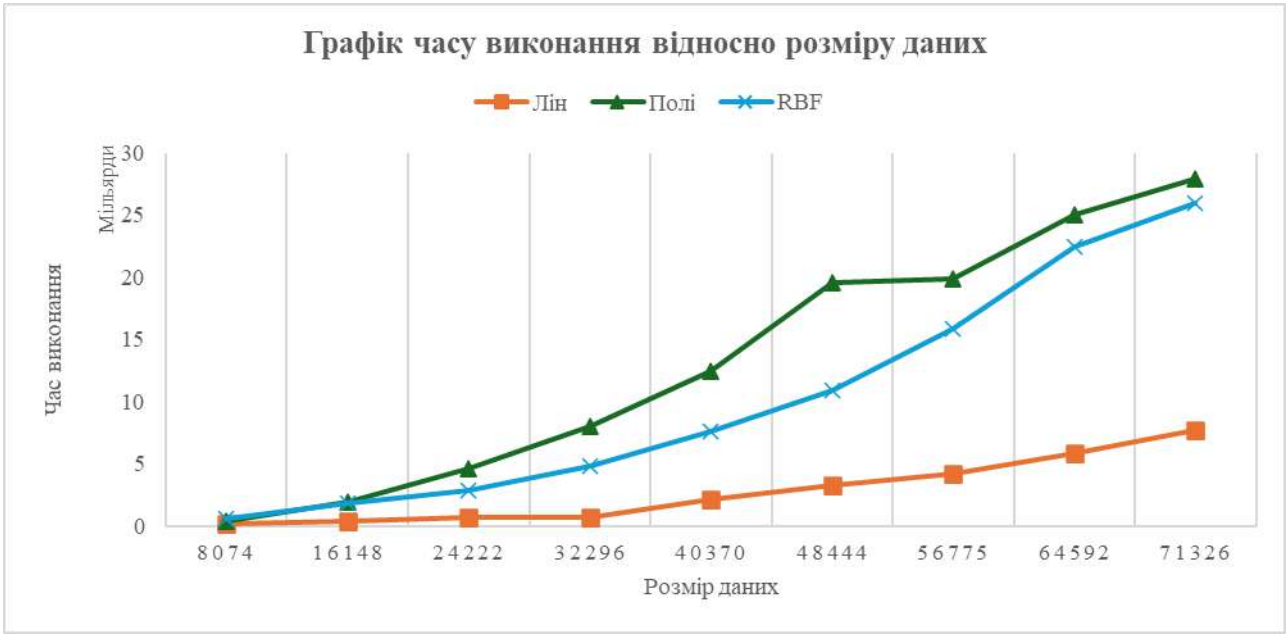


Рисунок 4.5 – Вбудоване програмне забезпечення для алгоритму SVM на основі СО: Графік часу виконання та розміру даних для набору даних-експериментів «Pulsar»

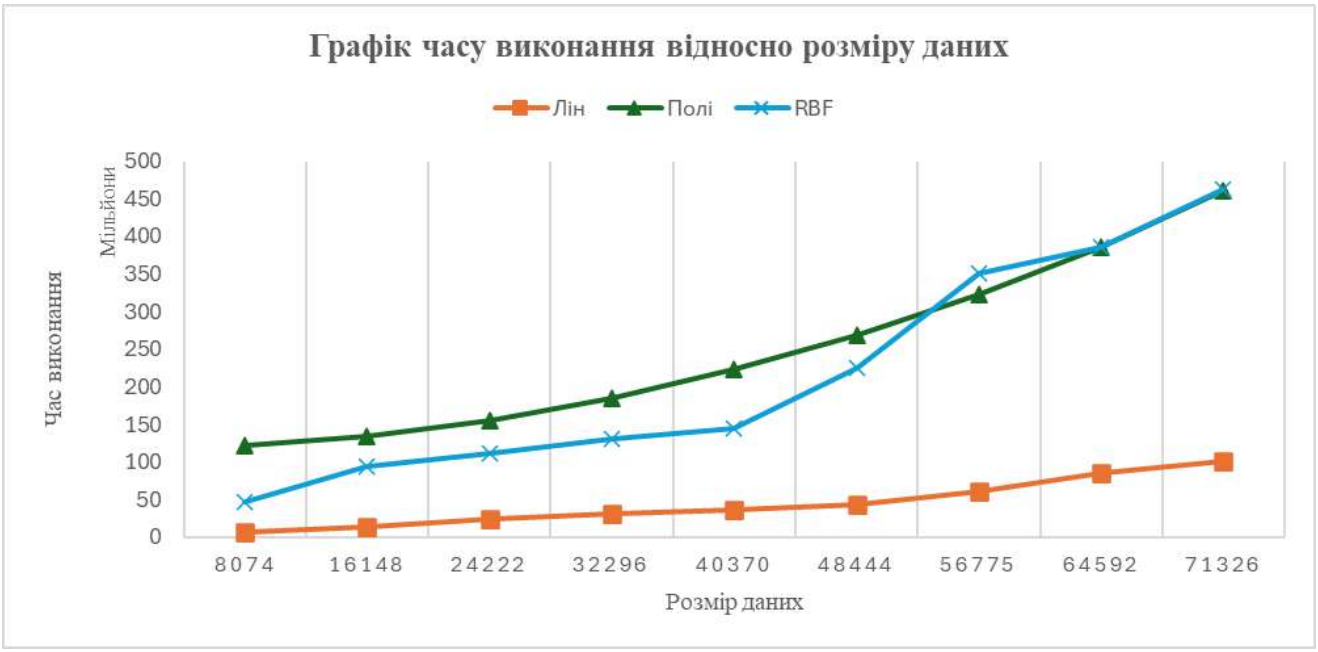


Рисунок 4.6 – Вбудоване апаратне забезпечення для алгоритму SVM на основі СО: Графік часу виконання та розміру даних для набору даних-експериментів «Pulsar»

4.4 Аналіз часу виконання з різною кількістю ітерацій

Попередньо описаний аналіз часу виконання отримано з різними розмірами даних та постійною кількістю ітерацій. Аналогічно аналізу точності, було проведено додаткові експерименти для аналізу часу виконання з різною кількістю ітерацій та постійним розміром даних. У цьому випадку було змінено максимальну кількість ітерацій від 500 до 3000 з кроком 500, щоб знайти мінімальне значення. Час виконання для вбудованого апаратного і програмного забезпечення також отримується з різною кількістю ітерацій для обох наборів даних-експериментів з розміром навчального набору даних 50%. Час виконання для вбудованого апаратного забезпечення для наборів даних-експериментів «Pulsar» та «Iris Flowers» представлено на Рисунках 4.7 і 4.8 відповідно. Візуально, як показано на Рисунку 4.7, для запропонованих вбудованих апаратних архітектур час виконання майже лінійно збільшується зі зростанням кількості ітерацій для всіх трьох ядер для наборів даних-експериментів «Pulsar». Для набору даних-експериментів «Iris Flowers», як показано на Рисунку 4.8, час виконання вбудованого апаратного забезпечення майже лінійно збільшується зі зростанням кількості ітерацій для лінійного та поліноміального ядер, в той час як для ядра RBF час виконання вбудованого апаратного забезпечення залишається таким самим зі зростанням кількості ітерацій.

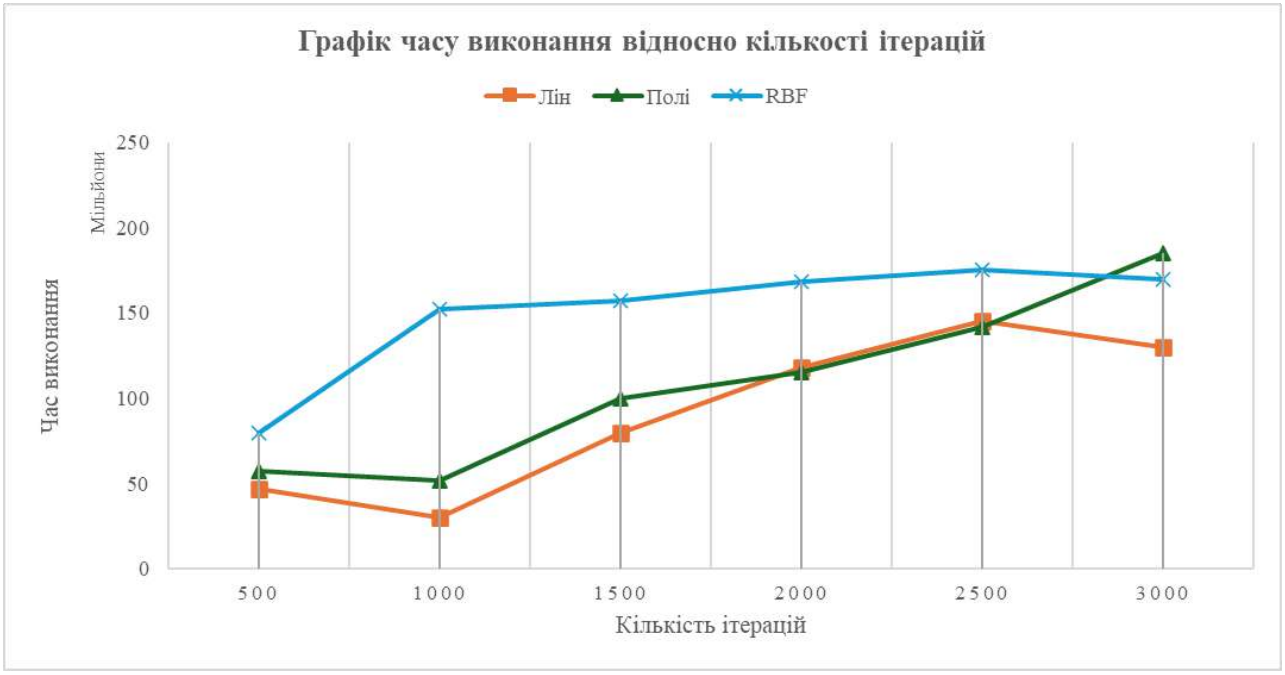


Рисунок 4.7 – Вбудоване апаратне забезпечення для алгоритму SVM на основі СО: Графік часу виконання та кількості ітерацій для набору даних-експериментів «Pulsar»

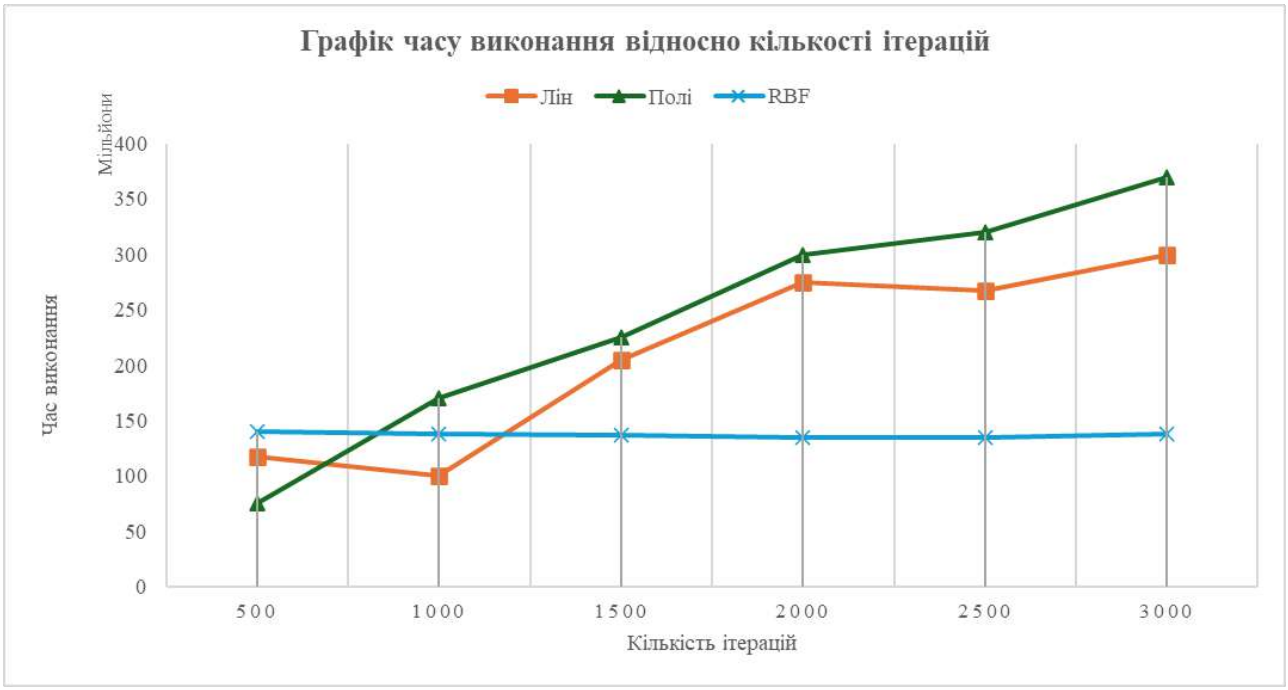


Рисунок 4.8 – Вбудоване апаратне забезпечення для алгоритму SVM на основі СО: Графік часу виконання та кількості ітерацій для набору даних-експериментів «Iris Flowers»

Для запропонованого вбудованого апаратного та програмного забезпечення було використано максимальну кількість ітерацій (1000 ітерацій) як поріг для знаходження мінімального значення, замість використання конкретного критерію зупинки. Таким чином, розв'язник оптимізації змушений досягнути максимальної кількості ітерацій, щоб завершити виконання алгоритму SVM на базі опуклої оптимізації. Більш того, критерій зупинки припиняє виконання алгоритму, коли функція об'єкту збігається до мінімального значення, що може або не може зменшити (або збільшити) загальний час виконання.

4.5 Аналіз прискорення

Приріст продуктивності (або прискорення), що виникає від вбудованого апаратного забезпечення порівняно з вбудованим програмним забезпеченням, що виконується на ARM Cortex-A9, для запропонованого алгоритму з використанням трьох різних математичних ядер, представлений у стовпці 6 у таблицях 4.1, 4.2, 4.3, 4.4, 4.6 та 4.6. Прискорення вимірюється за допомогою Рівняння (24). На Рисунках 4.9 та 4.10 показано прискорення відносно розмірів даних (відсоток навчального набору даних) для запропонованого вбудованого апаратного забезпечення для алгоритму SVM на базі опуклої оптимізації з лінійним, поліноміальним та гаусовим RBF ядрами для наборів даних-експериментів «Pulsar» та «Iris Flowers», відповідно. На перший погляд, як показано на Рисунках 4.9 та 4.10, прискорення зазвичай збільшується зі збільшенням відсотка навчального набору даних для обох наборів даних.

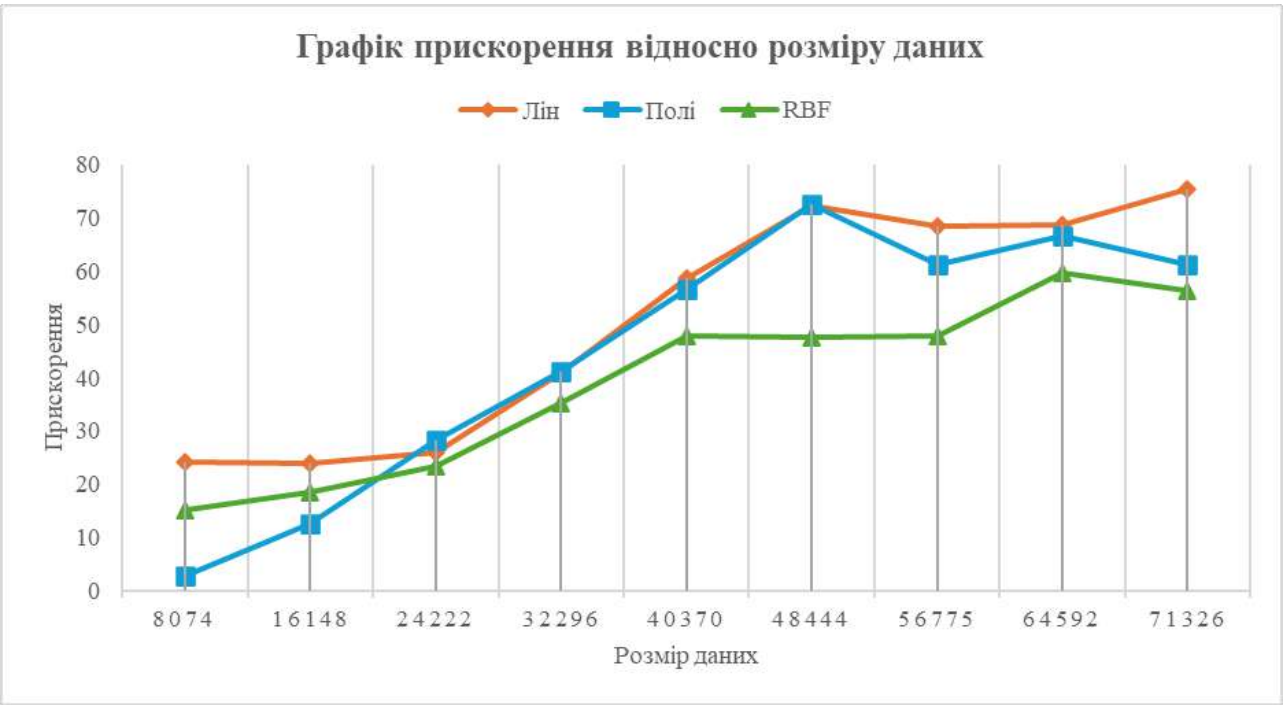


Рисунок 4.9 – Вбудоване програмне забезпечення для алгоритму SVM на основі СО: Графік прискорення та розміру даних за допомогою набору даних-експериментів «Pulsar»

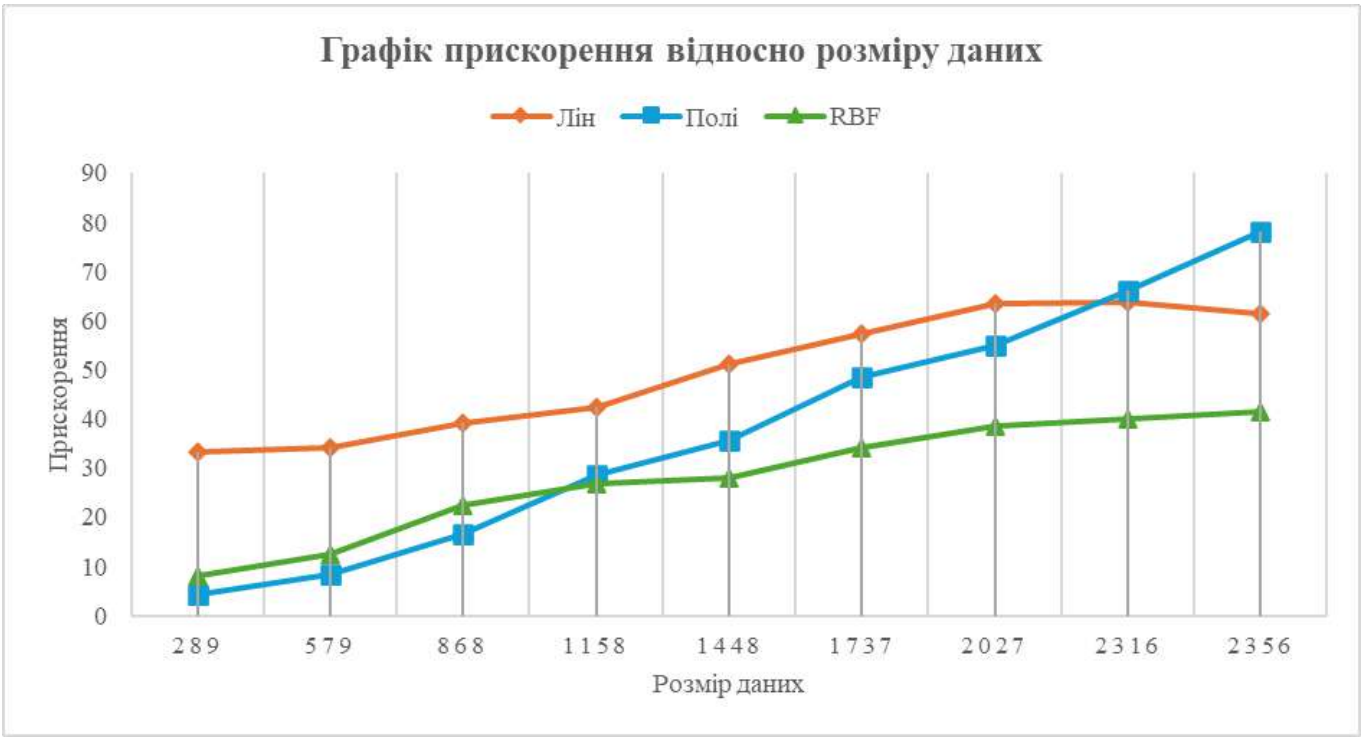


Рисунок 4.10 – Вбудоване апаратне забезпечення для алгоритму SVM на основі СО: Графік прискорення та розміру даних за допомогою набору даних-експериментів «Iris Flowers»

Для набору даних-експериментів «Iris Flowers», як на Рисунку 4.10 (і з Таблиць 4.4, 4.5 та 4.6), для набору даних з поліноміальним ядром прискорення зростає лінійно (від 4 разів до 79 разів швидше за програмні аналоги), коли відсоток навчального набору даних зростає від 10% до 90%; для набору даних з лінійним ядром прискорення майже лінійно зростає (від 32 разів до 62 разів швидше за програмні аналоги) при збільшенні відсотка навчального набору даних з 10% до 70%, і прискорення залишається тим самим (62 рази швидше), коли відсоток навчального набору даних зростає від 70% до 90%; для набору даних з ядром RBF прискорення також майже лінійно зростає (від 8 разів до 41 разу швидше за програмні аналоги) при збільшенні відсотка навчального набору даних з 10% до 90%.

Для набору даних-експериментів «Pulsar», як на Рисунку 4.9 (і з Таблиць 4.1, 4.2 та 4.3), для набору даних з поліноміальним ядром прискорення зростає лінійно (від 3 разів до 72 разів швидше за програмні аналоги), коли відсоток навчального набору даних зростає від 10% до 60%, і прискорення трохи зменшується, коли відсоток навчального набору даних зростає від 70% до 90%; для набору даних з лінійним ядром прискорення зростає лінійно (від 23 разів до 75 разів швидше за програмні аналоги) при збільшенні відсотка навчального набору даних з 10% до 60% і з 70% до 90%; для набору даних з RBF прискорення зростає лінійно (від 14 разів до 58 разів швидше за програмні аналоги) при збільшенні відсотка навчального набору даних з 10% до 60% і з 80% до 90%.

Як зазначалося раніше, додаткові експерименти з програмним забезпеченням виконані на настільному комп'ютері; отже, також було порівняно вбудовані апаратні рішення, які працюють на частоті 100 МГц на Altera Cyclone V FPGA, з базовим програмним забезпеченням на мові Python, що працює на процесорі Intel i7 з тактовою частотою 2,3 ГГц. У цьому випадку запропоновані вбудовані апаратні рішення досягли прискорення у 3,1 рази порівняно з реалізацією на Python для набору даних «Pulsar» з лінійним ядром; і запропоновані вбудовані апаратні рішення досягли прискорення у 34,8 разів порівняно з реалізацією на

Python для набору даних «Iris Flowers» з поліноміальним ядром. Ці результати додатково підтверджують ефективність запропонованих вбудованих апаратних рішень.

У підсумку, спостерігається, що для алгоритму SVM на основі CO зі збільшенням кількості вибірок (тобто, векторів) збільшується як точність, так і загальне прискорення. У цьому випадку, коли класифікатор CO-SVM має більше вибірок для навчання, це може призвести до виявлення складних паттернів та створення кращої розділяючої гіперплощини. Крім того, зі зростанням розміру матриць і складності обчислень/операцій, настроєні та оптимізовані конструкції можуть бути найкращим шляхом прискорення та покращення різних метрик продуктивності алгоритмів SVM на основі CO порівняно з традиційними платформами обчислень, такими як універсальні процесори.

Дане дослідження показує, що більшість існуючих робіт запропонували апаратні архітектури або для тестування, або для тренування, але не для обох. Крім того, більшість цих запропонованих апаратних архітектур не були загальними або параметризованими. Додатково, більшість цих архітектур не були розроблені з урахуванням вбудованих пристроїв. Ні одна з цих робіт не запропонувала системних архітектур та пов'язаних технік для забезпечення обробки за реальним часом застосунків машинного навчання. У зв'язку з цим існуючі роботи не повідомляли відповідну системну область і не враховували пов'язану затримку доступу до пам'яті при повідомленні часу/прискорення. В результаті не було можливості зробити прямі порівняння продуктивності з існуючими роботами з апаратними архітектурами на базі FPGA в опублікованій літературі. У підсумку, на основі даного дослідження, не було знайдено жодної подібної роботи до запропонованої в даному дослідженні в опублікованій літературі, яка б надала апаратні прискорювачі на базі FPGA для SVM на основі CO, особливо для вбудованих пристроїв, або подібної роботи, яка запропонувала б системні архітектури, що є невід'ємним для застосунків машинного навчання у реальних сценаріях.

4.6 Висновки

В результаті проведеного експериментального дослідження виявлено, що запропонований вбудований апаратний прискорювач для SVM методу на основі опуклої оптимізації виконувався до 79 разів швидше, ніж його програмний еквівалент, який працював на вбудованих мікропроцесорах.

Це значне покращення продуктивності було досягнуто за допомогою кількох технік оптимізації апаратного забезпечення, вбудованих у запропоновані архітектури, включаючи створення кастомізованих та оптимізованих архітектур за рахунок експлуатації вбудованого паралелізму та конвеєрної природи обчислень/завдань; проектування обчислень/завдань для перекриття з доступом до пам'яті; методів передачі даних пакетами та попереднього завантаження.

Запропоновані вбудовані рішення (як апаратні, так і програмні) досягли до 100% точності класифікації та кращої продуктивності (тобто, за постійної кількості ітерацій для пошуку мінімуму під час оптимізації, потребували набагато менше ітерацій) порівняно з кодом Python на настільних комп'ютерах.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень розроблено новий, спеціалізований та оптимізований апаратний прискорювач на основі FPGA для методу опорних векторів (SVM) на базі опуклої оптимізації (CO) на вбудованих платформах. Запропоновані вбудовані архітектури є універсальними, параметризованими та масштабованими. Таким чином, без зміни внутрішніх апаратних архітектур дані вбудовані рішення можуть бути використані для різних наборів даних різних розмірів, можуть бути виконані на різних вбудованих платформах, включаючи платформи з останніми FPGA, і можуть бути використані для лінійних/нелінійних розділяючих, багатовимірних наборів даних.

Запропоноване вбудоване апаратне та програмне рішення може бути переконфігуровано для вибору відповідного математичного ядра (із трьох: лінійного, поліноміального та гаусового радіального базисного функціоналу), в залежності від вимог певної програми машинного навчання.

В результаті визначено, що надаючи загальні та незалежні IP (інтелектуальні власності) для кожного етапу, запропонований метод може бути використаний для будь-якої програми машинного навчання або інших застосувань, які вимагають алгоритму SVM, і не обмежуються конкретним застосуванням у певній галузі.

У першому розділі досліджено можливості використання FPGA для аналізу даних. Було виявлено, що FPGA може ефективно використовуватися для реалізації алгоритмів машинного навчання, зокрема класифікаторів. Було проаналізовано різні класифікатори машинного навчання, їх переваги та недоліки. Зокрема, в результаті проведеного аналізу виявлено, що існуючі підходи до реалізації класифікаторів на базі FPGA мають такі недоліки, як обмежена швидкодія, обмежені можливості обробки складних алгоритмів та обмежені ресурси пристрою. Крім того, деякі підходи можуть бути недостатньо оптимізованими під конкретні завдання або дати менш точні результати через обмежену можливість роботи з нелінійними даними. Було виявлено, що використання відомих методів класифікаторів машинного навчання мають певні обмеження та недоліки. Зокрема,

використання лінійних класифікаторів для нелінійно роздільних даних може призводити до зниження точності результатів. Таким чином, при повній відсутності подібної роботи в опублікованій літературі, яка б надала апаратні архітектури на основі FPGA, особливо для SVM методу основаного на опуклій оптимізації на вбудованих пристроях, або подібної роботи, яка пропонує архітектури на рівні системи, що є невід'ємною для застосувань машинного навчання для сценаріїв реального світу, було виявлено потребу в розвитку оптимізованого та удосконаленого методу створення класифікатора на базі FPGA, який би ефективно використовував можливості цього пристрою та забезпечував високу точність результатів.

У другому розділі було досліджено та проаналізовано важливі аспекти оптимальної гіперплощини, яка є центральним елементом у методі опорних векторів. Виявлено, що оптимальна гіперплощина максимізує роздільну здатність між класами, забезпечуючи найбільш ефективне розділення даних у просторі. Під час аналізу було виявлено, що правильний вибір гіперплощини дозволяє підвищити точність та надійність класифікації для різноманітних наборів даних. Було досліджено методи нелінійної оптимізації, які використовуються для побудови нелінійних розділяючих гіперплощин у випадках, коли дані не є лінійно роздільними. Встановлено, що використання таких методів дозволяє ефективно вирішувати задачі класифікації нелінійних даних, що розширює можливості застосування методу опорних векторів на різні типи даних. Було досліджено опуклу оптимізацію як ключовий компонент методу опорних векторів. Виявлено, що використання опуклої оптимізації дозволяє ефективно знаходити глобальний мінімум функції втрат для оптимального розташування гіперплощини у просторі. Такий підхід забезпечує стійкість та надійність роботи методу опорних векторів навіть у випадку складних та нелінійних задач класифікації.

У третьому розділі було досліджено та проаналізовано основи створення SVM-класифікатора для аналізу даних на базі FPGA. Виявлено, що використання методу опорних векторів у поєднанні з опуклою оптимізацією дозволяє ефективно розв'язувати задачі класифікації навіть для складних та нелінійних наборів даних.

Архітектура на рівні системи була детально проаналізована з метою розробки оптимізованої системи аналізу даних на базі FPGA. Встановлено, що ефективне поєднання апаратних та програмних компонентів дозволяє досягти значного прискорення процесу аналізу даних та забезпечує високу швидкодію системи. Для реалізації запропонованого методу було створено вбудований апаратний дизайн з урахуванням специфіки завдань аналізу даних, що дозволило створити ефективний інструмент для реалізації SVM-класифікатора на основі опуклої оптимізації на FPGA. Виявлено, що використання апаратних ресурсів дозволяє значно прискорити процес аналізу даних та покращити загальну швидкодію системи.

У четвертому розділі в результаті проведеного експериментального дослідження було виявлено, що запропонований вбудований апаратний прискорювач для SVM методу на основі опуклої оптимізації виконувався до 79 разів швидше, ніж його програмний еквівалент, який працював на вбудованих мікропроцесорах. Це значне покращення продуктивності було досягнуто за допомогою кількох технік оптимізації апаратного забезпечення, вбудованих у запропоновані архітектури, включаючи створення кастомізованих та оптимізованих архітектур за рахунок експлуатації вбудованого паралелізму та конвеєрної природи обчислень/завдань; проектування обчислень/завдань для перекриття з доступом до пам'яті; методів передачі даних пакетами та попереднього завантаження. Запропоновані вбудовані рішення (як апаратні, так і програмні) досягли до 100% точності класифікації та кращої продуктивності (тобто, за постійної кількості ітерацій для пошуку мінімуму під час оптимізації, потребували набагато менше ітерацій) порівняно з кодом Python на настільних комп'ютерах.

У подальшому розроблений метод можна вдосконалити за рахунок можливості додати динамічні переконфігуровані апаратні прискорювачі [41] для застосувань у машинному навчанні з метою інтеграції розумних та адаптивних властивостей.

Розроблений програмний продукт може мати такі сфери застосувань машинного навчання, як медичне тестування для діагностики раку, аналіз даних

для контролю якості, класифікація зображень та розпізнавання мови, задовольняючи при цьому пов'язані обмеження вбудованих пристроїв.

За темою кваліфікаційної роботи магістра опублікована одна стаття у фаховому науковому журналі «Computer systems and information technologies» у 2024 році [1].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Лисенко С. М., Шпуляр Є. М. Method for creating SVM-classifier for data analysis on FPGA. Науковий журнал «Computer systems and information technologies», 2024.

2. Xiaohui Song, Hong Wang, Lingfeng Wang. FPGA Implementation of a Support Vector Machine Based Classification System and Its Potential Application in Smart Grid. 11th International Conference on Information Technology: New Generations. 2014

3. Frank Bruno, Guy Eschemann. The FPGA Programming Handbook: An Essential Guide to FPGA Design for Transforming Your Ideas into Hardware Using SystemVerilog and VHDL, Second Edition. 2024. 392.

4. Shereen Afifi, Hamid GholamHosseini, Roopak Sinha. A system on chip for melanoma detection using FPGA-based SVM classifier. Volume 65. 2019. Pages 57-68.

5. Frank Bruno. FPGA Programming for Beginners: Bring your ideas to life by creating hardware designs and electronic circuits with SystemVerilog. Packt. 2021. 368.

6. System Simulation, Architecture Exploration, Power Management, Performance Analysis. URL: https://www.mirabilisdesign.com/field-programmable-gate-arrays-fpgas/

7. В. Чумак, І. Свид. Створення модуля VHDL-опису при проектуванні цифрових систем на ПЛІС в Xilinx ISE Design Suite. // Перспективні напрямки сучасної електроніки, інформаційних і комп'ютерних систем (MEICS-2019). Тези доповідей на IV Всеукраїнській науково-практичній конференції: 27-29 листопада 2019 р., м. Дніпро. – Дніпро, Дніпровський національний університет імені Олеся Гончара, Кременчук: ПП Щербатих О. В., 2019. – С. 94-95

8. Iryna Svyd, Oleksandr Vorgul, Valerii Semenets, Oleg Zubkov, Valeriia Chumak, Natalia Boiko. Special Features of the Educational Component “Design of Devices on Microcontrollers and FPGA”. // II International Scientific and Practical Conference Theoretical and Applied Aspects of Device Development on Microcontrollers and FPGAs (MC&FPGA), Kharkiv, Ukraine, 2020, pp. 55-57.

9. Afef Saidi, Slim Ben Othman, Meriam Dhouibi, Slim Ben Saoud. FPGA-based implementation of classification techniques. Integration. Volume 81. 2021, Pages 280-299.

10. Prasanna, D.L., Tripathi, S.L. A Review on Tongue Based Assistive Technology, Devices and FPGA Processors Using Machine Learning Module. Wireless Pers Commun 134, 151–170 (2024).

11. H. Meng, K. Appiah, A. Hunter and P. Dickinson, "FPGA implementation of Naive Bayes classifier for visual object recognition," CVPR 2011 WORKSHOPS, Colorado Springs, CO, USA, 2011, pp. 123-128.

12. Laxmisagar, H.S., Hanumantharaju, M.C. FPGA implementation of breast cancer detection using SVM linear classifier. Multimed Tools Appl 82, 41105–41128. 2023

13. Y. Zhou, Z. Chen and X. Huang, "A system-on-chip FPGA design for real-time traffic signal recognition system," 2016 IEEE International Symposium on Circuits and Systems (ISCAS), Montreal, QC, Canada, 2016, pp. 1778-1781.

14. Babu, P., Parthasarathy, E. Reconfigurable FPGA Architectures: A Survey and Applications. J. Inst. Eng. India Ser. B 102, 143–156 (2021).

15. Afifi, S., GholamHosseini, H. & Sinha, R. FPGA Implementations of SVM Classifiers: A Review. SN COMPUT. SCI. 1, 133 (2020).

16. Amezzane, I., Fakhri, Y., El Aroussi, M., Bakhouya, M. (2020). Hardware Acceleration of SVM Training for Real-Time Embedded Systems: Overview. In: Dos Santos, S., Maslouhi, M., Okoudjou, K. (eds) Recent Advances in Mathematics and Technology. Applied and Numerical Harmonic Analysis. Birkhäuser, Cham.

17. Ghosh, A., Senthilrajan, A. Comparison of machine learning techniques for spam detection. Multimed Tools Appl 82, 29227–29254 (2023).

18. Rahhal Errattahi, Asmaa EL Hannani, Thomas Hain, Hassan Ouahmane. System-independent ASR error detection and classification using Recurrent Neural Network. Computer Speech & Language. Volume 55. 2019, Pages 187-199.

19. Lopes, F.F.; Ferreira, J.C.; Fernandes, M.A.C. Parallel Implementation on FPGA of Support Vector Machines Using Stochastic Gradient Descent. J. Electron. 2019, 8, 631.

20. Elgawi, O.; Mutawa, A.M.; Ahmad, A. Energy-Efficient Embedded Inference of SVMs on FPGA. In Proceedings of the 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Miami, FL, USA. 2019; pp. 164–168.

21. Koliogeorgi, K.; Zervakis, G.; Anagnostos, D.; Zompakis, N.; Siozios, K. Optimizing SVM Classifier Through Approximate and High Level Synthesis Techniques. In Proceedings of the 2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST), Thessaloniki, Greece. 2019; pp. 1–4.

22. Afifi, S.; Gholamhosseini, H.; Sinha, R. A system on chip for melanoma detection using FPGA-based SVM classifier. Elsevier J. Microprocess. Microsyst. (MICPRO) 2019, 65, 57–68.

23. Tsoutsouras, V.; Koliogeorgi, K.; Xydis, S.; Soudris, D. An Exploration Framework for Efficient High-Level Synthesis of Support Vector Machines: Case Study on ECG Arrhythmia Detection for Xilinx Zynq SoC. J. Signal Process. Syst. 2017, 88, 127–147.

24. Noronha, D.H.; Torquato, M.F.; Fernandes, M.A.C. A Parallel Implementation of Sequential Minimal Optimization on FPGA. Elsevier J. Microprocess. Microsyst. (MICPRO) 2019, 69, 138–151.

25. Vranjković, V.; Struharik, R. Coarse-grained reconfigurable hardware accelerator of machine learning classifiers. Proceedings of the IEEE International Conference on Systems Signals and Image Processing (IWSSIP'16), Bratislava, Slovakia. 2016; pp. 1–5.

26. Papadonikolakis, M.; Bouganis, C.-S. A novel FPGA-based SVM classifier. Proceedings of the 2010 International Conference on Field-Programmable Technology, Beijing, China. 2010; pp. 283–286.

27. Jim Ledin. Architecting High-Performance Embedded Systems: Design and build high-performance real-time digital systems based on FPGAs and custom circuits. Packt. 2021. 376.

28. Ross K. Snider. Advanced Digital System Design using SoC FPGAs: An Integrated Hardware/Software Approach 1st ed. 2023 Edition. Springer; 1st ed. 2023 edition. 2023. 455.

29. Robert Dunne. Introduction to X86 Machine Code Assembly Language: Using an FPGA with Verilog. Gaul Communications. 2023. 334.

30. Piccialli, V.; Sciandrone, M. Nonlinear optimization and support vector machines. 4OR 2018, 16, 111–149.

31. Juan Jose Rodriguez Andina, Eduardo de la Torre Arnanz, Maria Dolores. FPGAs Fundamentals, Advanced Features, and Applications in Industrial Electronics. Valdes. ISBN 9780367656249. 265 Pages. Published September. 2020. CRC Press.

32. Chao Wang. Domain-Specific Computer Architectures for Emerging Applications. Machine Learning and Neural Networks. ISBN 9780367374532. 416 Pages 64 Color & 142 B/W Illustrations. 2024. Chapman & Hall.

33. Xavier Savarimuthu, SJ, Sivakannan Subramani, Alex Noel Joseph Raj. Artificial Intelligence for Multimedia Information Processing. Tools and Applications. ISBN 9781032521473. 346 Pages 69 B/W Illustrations. 2024. CRC Press.

34. DE1-SoC User Manual (rev.F/rev.G Board) Terasic Technology Inc. URL: https://www.terasic.com.tw/cgi-bin/page/archive.pl

35. Shahrouzi, S.N.; Perera, D.G. Optimized hardware accelerators for data mining applications on embedded platforms: Case study principal component analysis. Microprocess. Microsyst. 2019, 65, 79–96.

36. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B. Scikit-learn: Machine Learning in Python. J. Mach. Learn. Res. 2011, 12, 2825–2830.

37. Nilsson, P.; Shaik, A.U.R.; Gangarajaiah, R.; Hertz, E.; Nilsson, P. Hardware implementation of the exponential function using Taylor series. NORCHIP; IEEE: Piscataway, NJ, USA, 2014; pp. 1–4

38. Sandeep Saini, Kusum Lata, G.R. Sinha. VLSI and Hardware Implementations using Modern Machine Learning Methods. ISBN 9781032061719. 328 Pages 119 B/W Illustrations. 2021. CRC Press.

39. Fisher,R. A.. (1988). Iris. UCI Machine Learning Repository. URL: https://doi.org/10.24432/C56C76.

40. Lyon,Robert. (2017). HTRU2. UCI Machine Learning Repository. URL: https://doi.org/10.24432/C5DK6R.

41. Shahrouzi, S.N.; Perera, D.G. Dynamic partial reconfigurable hardware architecture for principal component analysis on mobile and embedded devices. EURASIP J. Embed. Syst. 2017, 2017, 212.

42. Mounir Maaref. Architecting and Building High-Speed SoCs: Design, develop, and debug complex FPGA-based systems-on-chip. Packt Publishing. 2022. 426.

43. Адель Мелліт, Беккай Хаджі, Лубна Бусселам. A Practical Guide for Simulation and FPGA Implementation of Digital Design. Springer. 2023. 328.

44. Blaine C. Readler. Verilog by Example: A Concise Introduction for FPGA Design First Edition. Lightning Source Inc; First Edition. 2011. 114.

45. Pong P. Chu. FPGA Prototyping by VHDL Examples 1st Edition. Wiley-Interscience; 1st edition. 2008. 468.

46. Simon Monk. Programming FPGAs: Getting Started with Verilog 1st Edition. McGraw Hill TAB; 1st edition. 2016. 192.

47. Justin Rajewski. In O'Reilly Media. Learning FPGAs: Digital Design for Beginners with Mojo and Lucid HDL 1st Edition. 1st edition. 2017. 225.

48. Kindle Edition. Juan José Rodriguez Andina. FPGAs: Fundamentals, Advanced Features, and Applications in Industrial Electronics 1st Edition. CRC Press; 1st edition. 2017. 368.

49. A. Arockia Bazil Raj. FPGA-Based Embedded System Developer's Guide. ISBN 9781498796750. 846 Pages 292 B/W Illustrations. 2018. CRC Press.

50. Hartmut F.-W. Sadrozinski, Jinyuan Wu. Applications of Field-Programmable Gate Arrays in Scientific Research. ISBN 9781138112483. 170 Pages 2 Color & 91 B/W Illustrations. 2017 by CRC Press.

51. Pierre-Emmanuel Gaillardon. Reconfigurable Logic, Architecture, Tools, and Applications. ISBN 9781482262186. 554 Pages 20 Color & 216 B/W Illustrations. Published December 10, 2015. CRC Press.

52. Daniel Firestone et al. “Azure Accelerated Networking: Smart- NICs in the Public Cloud.” 15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Ren- ton, WA, USA, 2018. 2018, pp. 51–66.

53. Carsten Heinz, Yannick Lavan, Jaco Hofmann, and Andreas Koch. “A Catalog and In-Hardware Evaluation of Open-Source Drop-In Compatible RISC-V Softcore Processors.” IEEE Proc. International Conference on ReConFigurable Computing and FPGAs (ReConFig). IEEE. 2019.

54. J. Gray. “GRVI Phalanx: A Massively Parallel RISC-V FPGA Accelerator Accelerator.” IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). 2016, pp. 17–20. doi: 10.1109/FCCM.2016.12.

55. Jeffrey Fong, Xiang Wang, Yaxuan Qi, Jun Li, and Weirong Jiang. ParaSplit: A scalable architecture on FPGA for terabit packet classification: 20th Annual Symposium on High-Performance Interconnects. IEEE. 2012, pp. 1–8.

56. Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H. Anderson, Stephen Brown, and Tomasz Czajkowski. LegUp: High-level Synthesis for FPGA- based Processor/Accelerator Systems.In: Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays. FPGA ’11. Monterey, CA, USA: ACM, 2011, pp. 33–36. isbn: 978-1-4503-0554-9. doi: 10.1145/1950413.1950423. URL: http://doi.acm.org/10.1145/1950413.1950423.

57. M. Hutton, V. Betz, J. Anderson, FPGA synthesis and physical design, in Electronic Design Automation for IC Implementation, Circuit Design, and Process Technology. CRC Press, 2017, pp. 395–436

58. R. Kastner, J. Matai, S. Neuendorffer, Parallel programming for FPGAs 2018. ArXiv e-prints arXiv:1805.03648

59. S.A. Edwards, R. Townsend, M.A. Kim, Compositional dataflow circuits. Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design (Vienna, 2017), pp. 175–184.

60. Simplilearn Inc. Machine Learning: What It Is and Why It Matter. February 2019. URL: https://www.simplilearn.com/what-is-machine-learning-and-why-it-matters-article.

61. Mohsin, M.A.; Perera, D.G. An FPGA-Based Hardware Accelerator for K-Nearest Neighbor Classification for Machine Learning on Mobile Devices. Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies, Toronto, ON, Canada, 6–7 June 2018; pp. 1–7.

62. Piccialli, V.; Sciandrone, M. Nonlinear optimization and support vector machines. 4OR 2018, 16, 111–149.

63. Miro, J.P. FPGA-Based Accelerators for Convolutional Neural Networks on Embedded Devices. Master's Thesis, Department of Electrical & Computer Engineering, University of Colorado Colorado Springs, Colorado Springs, CO, USA, 2020.

64. Ramadurgam, S. Optimized Embedded Architectures and Techniques for Machine Learning Algorithms for On-Chip AI Acceleration. Ph.D. Thesis, Department of Electrical & Computer Engineering, University of Colorado Colorado Springs, Colorado Springs, CO, USA, 2021.

65. Raghavan, R.; Perera, D.G. A Fast and Scalable FPGA-Based Parallel Processing Architecture for K-Means Clustering for Big Data Analysis. Proceedings of the IEEE Pacific Rim International Conference on Communications, Victoria, BC, Canada, 2017; pp. 1–8.

66. Ajao L. A. et al. Learning of embedded system design, simulation and implementation: a technical approach. American Journal of Embedded Systems and Applications. – 2016. – Т. 3. – №. 3. – С. 35-42.

67. Zurawski R. Embedded Systems Handbook: Embedded systems design and verification. – CRC press, 2018.

68. Suzuki K. et al. CFRP: A functional reactive programming language for small-scale embedded systems. Theory and Practice of Computation (WCTP 2016). – 2017.

69. Manor E., Greenberg S. Efficient Hardware/Software partitioning for Heterogeneous Embedded Systems. IEEE International Conference on the Science of Electrical Engineering in Israel (ICSEE). – IEEE, 2018. – С. 1-4.

70. Aysu A., Schaumont P. Hardware/software co-design of physical unclonable function based authentications on FPGAs Microprocessors and Microsystems. – 2015. – Т. 39. – №. 7. – С. 589-597.

71. Iguider A. et al. Heuristic approach for multi-objective hardware/software partitioning. 19th IEEE Mediterranean Electrotechnical Conference (MELECON). – IEEE, 2018. – С. 209-212.

72. Campbell K. et al. Debugging and verifying SoC designs through effective cross-layer hardware-software co-simulation Proceedings of the 53rd Annual Design Automation Conference. – ACM, 2016. – С. 7.

73. Hatcliff J. et al. Model-Based Development for High-Assurance Embedded Systems. International Symposium on Leveraging Applications of Formal Methods. – Springer, Cham, 2018. – С. 539-545.

74. Abdelfattah M. S., Betz V. T. Field Programmable Gate-Array with Embedded Network-on-Chip Hardware and Design Flow : 14060253 США. – 2015.

75. Filali K., Nouri K., Lahbib Y. Soft computing programmable control systems-on-chip: Trends and surface polynomial modeling International Conference on Control, Decision and Information Technologies (CoDIT). – IEEE, 2016. – С. 718-723.

76. Martin G., Cavalli D., Firmin F. Embedded FPGA with multiple configurable flexible logic blocks instantiated and interconnected by abutment : 10116311 США. – 2018.

77. Lee J. H., Kim K. M. System-on-chip (SoC) devices, display drivers and SoC systems including the same : 10008182 США. – 2018.

78. Langhammer M., Baeckler G. High Density and Performance Multiplication for FPGA 2018 IEEE 25th Symposium on Computer Arithmetic (ARITH). – IEEE, 2018. – С. 5-12.

79. Khalili F., Procaccini M., Giorgi R. Reconfigurable Logic Interface Architecture for CPU-FPGA Accelerators HiPEAC ACACES. – 2018.

ДОДАТОК А. Стаття
(обов'язковий)

S. LYSENKO, Y. SHPULIAR
Khmelnytskyi National University, Khmelnytskyi, Ukraine

METHOD FOR CREATING SVM CLASSIFIER FOR DATA ANALYSIS ON FPGA

The paper explores the use of SVM classifier method for data analysis on FPGA, which, despite its effectiveness, may face challenges related to limited resources and data processing speed. In this context, there is a need to develop new methods for integrating SVM classifiers with high-performance computing hardware. The increasing demand for speed and energy efficiency requires new approaches to implementing machine learning methods. One of the key tools for data classification and analysis is the Support Vector Machine (SVM), widely used in business, science, medicine, and many other fields. Developing an efficient and optimized method for creating SVM classifiers for FPGA requires further research and development, as existing methods may be suboptimal in terms of speed and FPGA resource utilization. The article provides an overview of known hardware solutions to this problem, proposed in the current scientific literature. Additionally, the effectiveness of combining hardware and software components to achieve significant acceleration of the data analysis process is discussed. The article emphasizes the need for further research and improvement to fully realize the transformative potential of machine learning classification methods.

Keywords: SVM classifier, machine learning, convex optimization, FPGA, data analysis methods, data classification, support vector machine..

1 Introduction

In today's world, the volume and complexity of data is growing, which requires effective methods of their analysis and processing. Machine learning, particularly classification methods, is becoming a key tool for automated data processing and analysis in fields ranging from medicine to finance. However, the growing need for speed and energy efficiency requires new approaches to the implementation of these methods. One of the key tools for data classification and analysis is the method of support vectors (SVM - Support Vector Machine), which is widely used in business, science, medicine and many other fields [1,3].

The use of the SVM-classifier creation method for FPGA-based data analysis is an urgent problem, since FPGAs provide high speed and energy efficiency, which makes them attractive for application in the field of machine learning [2,4]. Despite its effectiveness, the implementation of the SVM classifier can face challenges related to limited resources and data processing speed. In this context, there is a need to develop new methods of integrating SVM classifiers with high-performance computing equipment [3].

The study is devoted to the development of a method for creating an SVM classifier for data analysis based on FPGAs, known for their high speed and the possibility of parallel data processing, which makes them an ideal candidate for the effective implementation of machine learning classifiers.

2 Known methods of implementing FPGA classifiers

Methods of implementing classifiers on FPGAs include various techniques and approaches to implementing classification algorithms on programmable logic gate arrays (FPGAs).

In [6], an energy-efficient embedded architecture of a binarized SVM on an FPGA was proposed and implemented. Computational cores were developed in C/C++ programming languages and converted to HDL using Xilinx HLS high-level synthesis tools. The hardware design was performed on a Xilinx Virtex-6/7 FPGA. Results on FPGA performance metrics (especially speed and power) were compared to CPU and graphics processing unit (GPU) performance. From the obtained results, it can be seen that the FPGA and GPU have achieved a significant increase in speed compared to the CPU. However, the power consumption of the GPU was significantly higher than that of the FPGA. These results show that FPGA-based hardware architectures for SVM can achieve better performance, so they are suitable for embedded devices with tight power requirements.

In [8], an FPGA-based hardware design for an SVM classifier was proposed. In this case, three variable-size SVM models were implemented using different optimization methods. The proposed hardware was developed using the Xilinx Vivado HLS tool and implemented on the Xilinx Zynq7 ZC702 board. Results were obtained and presented in terms of area, acceleration, power, and classification accuracy. In addition, in this paper, the training phase was performed autonomously using the software; therefore, the support vectors were precomputed and passed to the proposed hardware, which is built only for the testing phase. This can be a problem for real-time machine learning applications.

In [10], an FPGA-based hardware-software design was proposed to accelerate the SVM algorithm by using a two-level approach: first optimize the global SVM structure and then refine it through design research. The proposed architecture was developed using the Xilinx Vivado HLS tool and implemented on the Xilinx Zynq Zedboard. Results were obtained for area, delay time, and acceleration. However, classification accuracy results were not reported. As mentioned, for high values of the SVM parameters, resource utilization (i.e. occupied area) increases significantly, which can be a problem for embedded devices with limited area requirements.

In [12], a scalable FPGA-based architecture was proposed to accelerate SVM classification. The hardware was designed in VHDL and implemented on an Altera Stratix III EP3SE260 board. Results were obtained and presented for acceleration; however, the occupied space was not released. In addition, a detailed discussion and analysis of the classification accuracy results is not provided. In this paper, only hardware was proposed for the testing phase. Thus, the reference vectors were precomputed and stored in the on-board memory for further processing during the testing phase.

3 A method of creating an SVM classifier for data analysis based on FPGA

An improved method of creating an SVM classifier for data analysis is proposed, which, unlike the known ones, performs data analysis using the hardware capabilities of the FPGA, which made it possible to reduce the time of data analysis, increase accuracy, and reduce software and hardware resources.

All embedded hardware and software studies were performed on the DE1-SoC Board development platform using the Altera Cyclone V SoC 5CSEMA5F31C6 device.

The appearance of the board for the development of DE1-SoC from Terasic is presented in Fig. 1 [13].

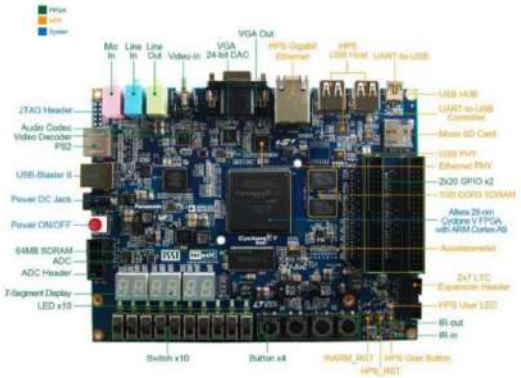


Figure 1 – Top view of the DE1-SoC development board

Figure 2 shows the system architecture for embedded hardware and software projects.

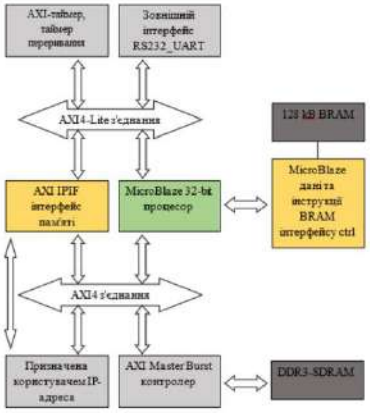


Figure 2 – System-level architecture

One of the goals of this design was to create a system-level architecture to train and classify a continuous stream of data using the AXI4 burst/stream interface. In a real-world application scenario, this feature allows for a direct connection between a user's IP hardware and a camera (for example, in an autonomous car) to process input data in real-time. This allows a given hardware IP to dynamically perform training and classification processes to adapt to an ever-changing environment. As shown in Figure 2, the AXI4-lite and AXI4 interfaces act as the "glue" logic for the entire system, including the ARM Cortex-A9 processor, internal peripherals, and user-defined IP. The AXI4-lite is a one-time transactional bidirectional interface, with memory mapped to the memory region.

During preload mode (Figure 3), the custom logic module determines the total number of bytes to be loaded using the above SVM specifications provided to the ARM Cortex-A9 via the work registers. After the training process

is completed, the AXI master burst controller is automatically configured for a write operation to save the weight vectors and offset values to DDR3-SDRAM for further computation/analysis.

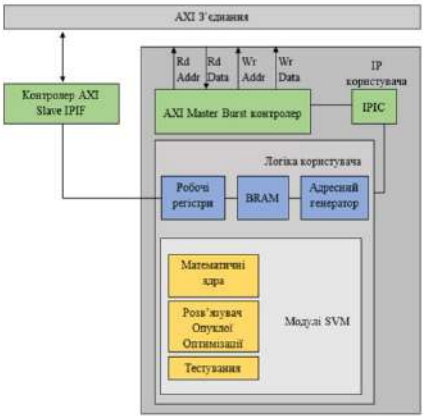


Figure 3 – Preload method and upper level architecture.

To develop the software for this SVM algorithm based on convex optimization, C++ was used with the help of Microsoft Visual Studio development tools. The obtained results are compared and verified with the Python open code results obtained from [14] (i.e. to check the correctness of functionality and operation as well as accuracy).

After analyzing the functional flow of the SVM algorithm based on convex optimization, it was determined that this complex algorithm should be divided into three stages (SVM Module in Fig. 3) to simplify the design process. The three steps of the CO-based SVM algorithm are performed non-sequentially to take advantage of the parallel processing characteristic of FPGA-based hardware [15].

At the first stage of this hardware project, an appropriate mathematical core was selected and implemented. Linear-inseparable vectors in the input space can be transformed into linear-separable vectors in the feature space by mapping data points to a higher-dimensional space [16]. This transformation can be performed using mathematical kernels (including linear, polynomial, and Gaussian kernels). To use mathematical kernels, the mapping function ($\Phi(x)$) in Equation (1) must satisfy the Mercer condition, which states that the dot product of two input vectors must be defined for all features as shown in Equation (1).

$$K(A_x, A_y) = \Phi(A_x) \times \Phi(A_y)$$
$$\int\int g(A_x)K(A_x, A_y)g(A_y)dA_x dA_y \geq 0 \quad (1)$$

The optimization step is the most complex operation among the three steps of the SVM algorithm based on convex function (CO) optimization. To reduce the complexity, step 2 is divided into three phases: parameter initialization, convex optimization, and displacement value computation. In this step, a dual form of SVM is used to formulate a general convex function optimization.

During the parameter initialization phase of stage 2, several parameters are calculated, including feature function parameters (ie, N, u), constraint parameters (ie, G, h, D, zconst), and other parameters (α, Gr, admissible point) .

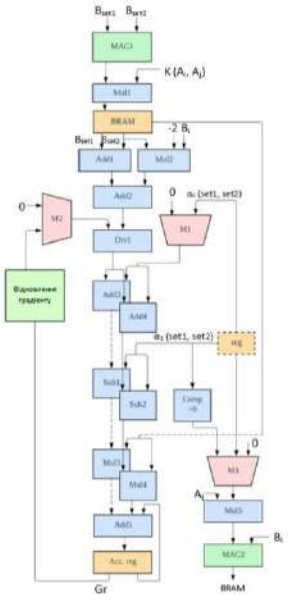


Figure 4 – Data path for Convex Optimization

After the parameter initialization phase, the optimization phase is performed. Five operations are performed during the optimization phase. The data processing scheme for optimization is shown in Figure 4, which consists of several adders, multipliers, subtractors, MAC modules, dividers, storage registers, comparators, and multiplexers. The optimization process consists in finding the minimum value of α. In this case, the value of α is calculated using modules Add1 to Add4, Mul1, Div1 and M1 (multiplexer)

The last step of stage 2 is the displacement value calculation phase. The z-shift value is also known as the displacement value, which represents the intersection of the hyperplane from the datum. The displacement value is calculated according to the following equation (2):

$$z = -\frac{1}{2}\left[\max_{\{x|B_y=-1\}}\left(\sum_{y=1}^{m}\alpha_x BK(A_x, A_y)\right) + \min_{\{x|B_y=+1\}}\left(\sum_{y=1}^{m}\left(\alpha_x B_x K(A_x, A_y)\right)\right)\right] \quad (2)$$

$$z = \frac{\left(\sum_{x=1}^{m} B_x - \sum_{x=1}^{m}\alpha_x B_{sv} K(A_x, A_y)\right)}{\text{кількість опорних векторів}, n_{sv}} \quad (3)$$

Stage 3 is the testing process. Usually, for classification, the input data set is divided into two samples: training and test. During phases 1 and 2, training is performed using the training set, while during phase 3, testing is performed using the test set. For the testing process, the input data vectors are classified into the class -1 (minus one) or +1 (plus one) based on the sign value of the function f (A). As shown in Figure 5, the data path for testing (classification) includes a multiplier, a MAC module, an adder, a multiplexer, and a comparator.

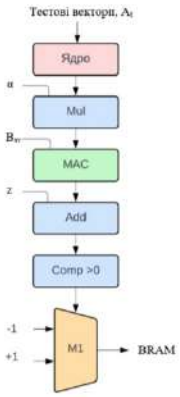


Figure 5. Data path for the test process.

4 Experiments

Testing was conducted to evaluate the feasibility and effectiveness of the proposed embedded hardware and software for a convex optimization (CO)-based SVM machine learning algorithm in terms of speed (speedup), accuracy, and scalability for handling different datasets of different sizes.

Execution time analysis. To evaluate the performance of the proposed embedded hardware, an embedded software for the CO-based SVM method was designed and implemented. The firmware runs on an ARM Cortex-A9 processor on the same DE1 Soc development platform. Execution times for embedded hardware and software designs of the CO-based SVM algorithm using linear, polynomial, and Gaussian RBF kernels are presented in Figures 6 and 7, respectively, for the Pulsar experimental datasets.

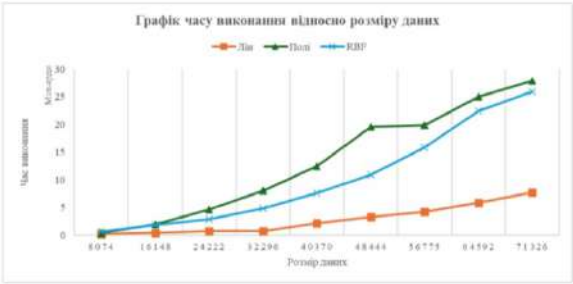


Figure 6 – Embedded software for CO-based SVM algorithm: Plot of execution time and data size for the Pulsar experimental data set

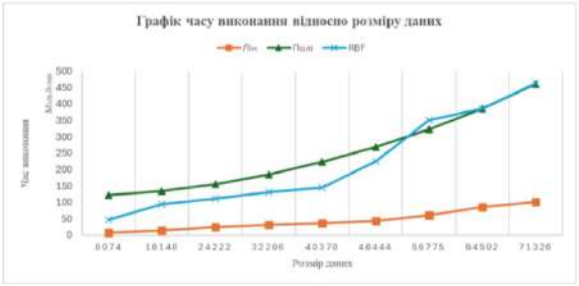


Figure 7 – Embedded hardware for the CO-based SVM algorithm: Execution time versus data size plot for the Pulsar experimental data set

Analysis of classification accuracy. As a result of the study, it can be noted that the classification accuracy varies depending on different data sets and different percentages of training sets. The Pulsar test dataset achieves the best classification accuracy of 96% with the polynomial kernel when the percentage of the training set is 90% of the dataset.

In addition to the proposed embedded hardware and software architectures, classification accuracy experiments were also performed on Python code running on a desktop computer. Plots of the accuracy results of the Python code design using linear, polynomial, and Gaussian RBF kernels are presented in Figures 8 and 9, respectively, for the Pulsar and Iris Flowers experimental datasets.

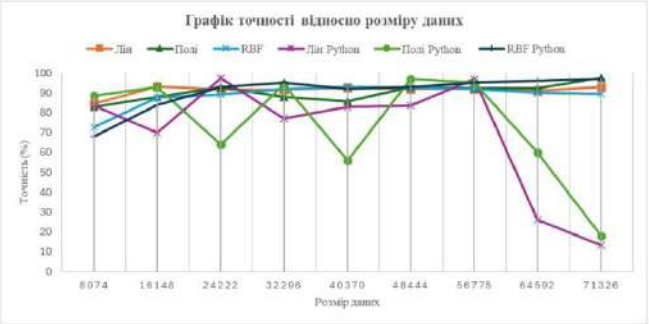


Figure 8 – Plot of classification accuracy and data size for the Pulsar experimental dataset

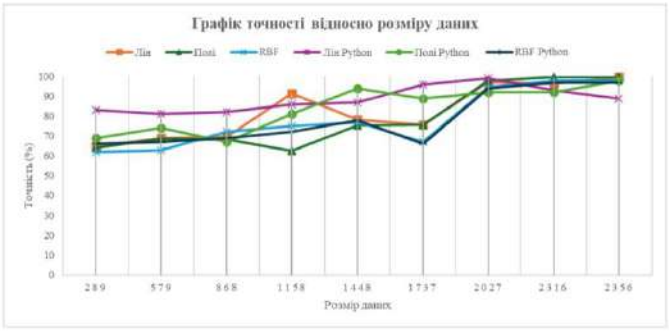


Figure 9 – Plot of classification accuracy and data size for the "Iris Flowers" experimental dataset

Conclusions

У work, based on the results of theoretical and practical research, a new, specialized and optimized hardware accelerator based on FPGA was developed for the support vector method (SVM) based on convex optimization (CO) on embedded platforms. The proposed embedded architectures are universal, parameterized, and scalable. Thus, without changing the internal hardware architectures, these embedded solutions can be used for different data sets of different sizes, can be implemented on different embedded platforms, including platforms with the latest FPGAs, and can be used for linear/nonlinear discriminating, multidimensional data sets. The proposed built-in hardware and software solution can be reconfigured to select the appropriate mathematical kernel (out of three: linear, polynomial, and Gaussian radial basis functional), depending on the requirements of a particular machine learning program.

This significant performance improvement was achieved through several hardware optimization techniques built into the proposed architectures, including creating customized and optimized architectures by exploiting the built-in parallelism and pipelined nature of computations/tasks; design calculations/tasks to overlap with memory access; methods of packet data transfer and preloading.

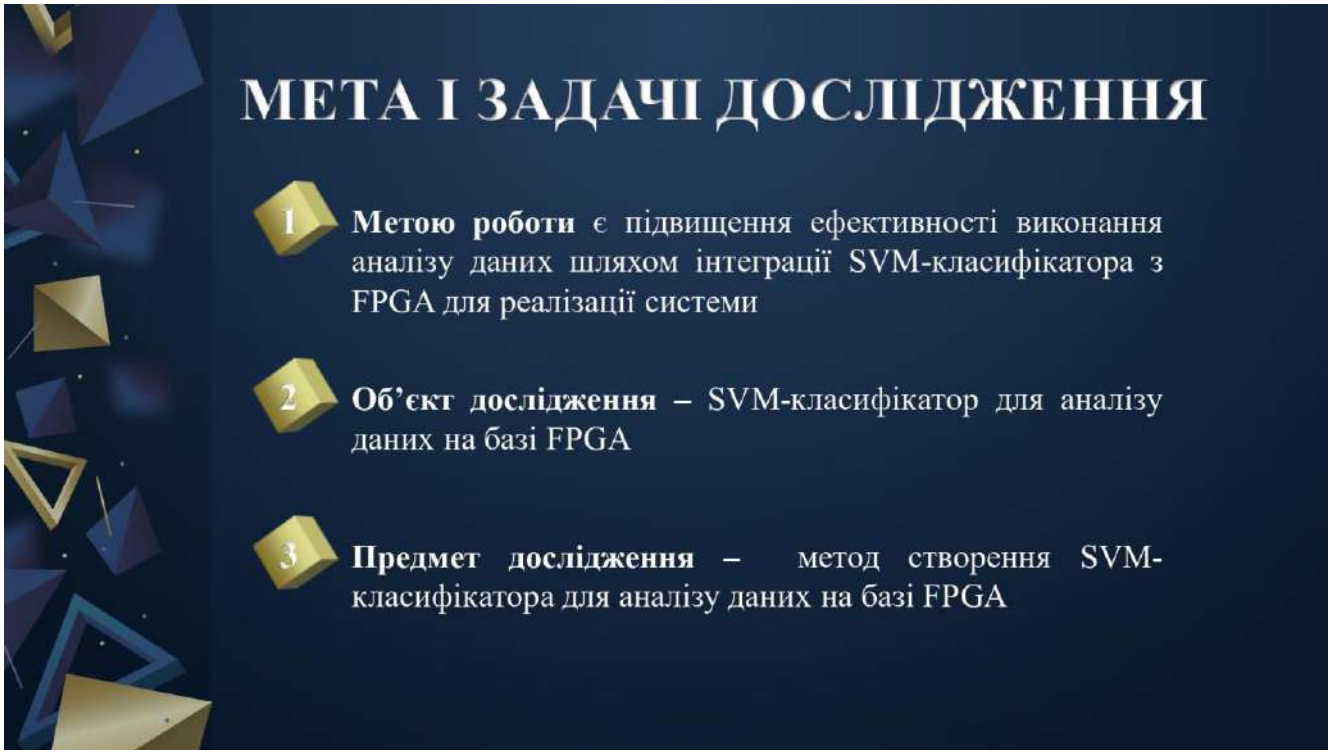
In the future, the developed method can be improved due to the possibility of adding dynamic reconfigurable hardware accelerators for applications in machine learning in order to integrate intelligent and adaptive properties.

The developed software can have machine learning applications such as medical testing for cancer diagnosis, data analysis for quality control, image classification and speech recognition, while satisfying the associated limitations of embedded devices.

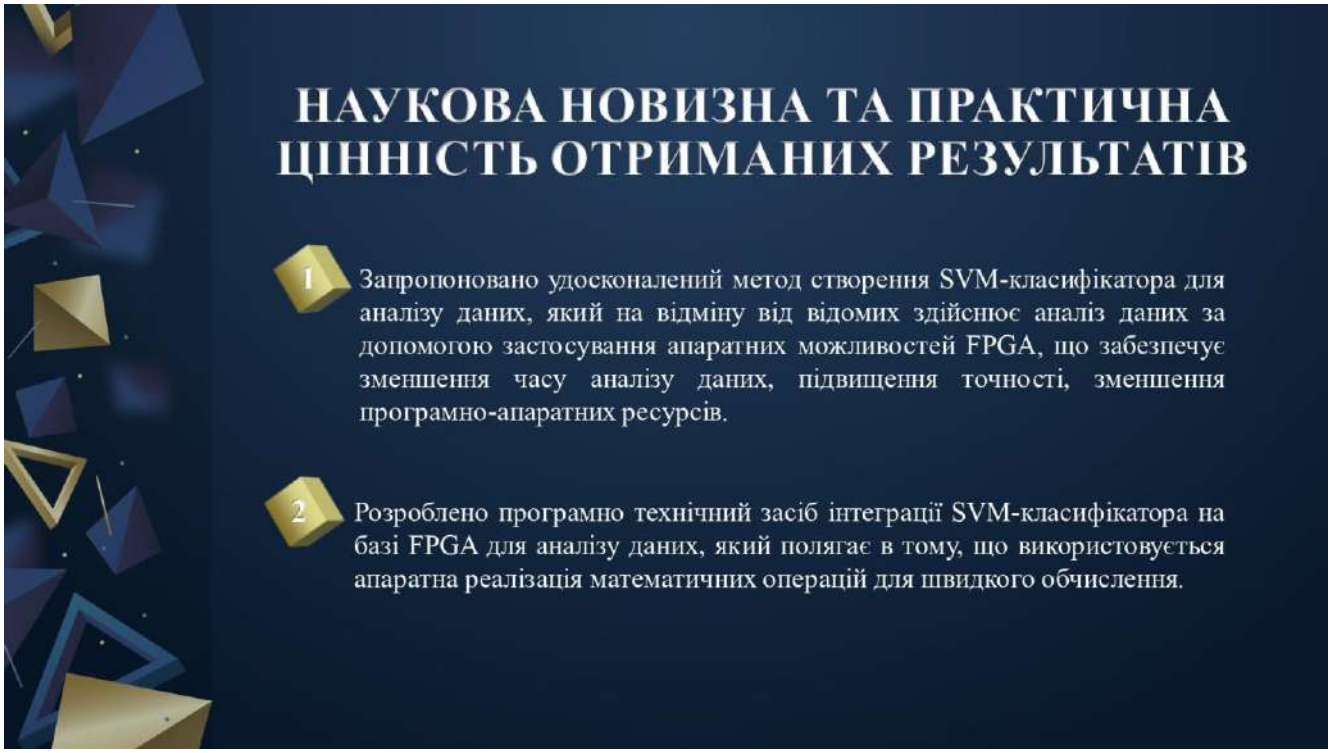
References

1. Afifi, S., GholamHosseini, H. & Sinha, R. FPGA Implementations of SVM Classifiers: A Review. SN COMPUT. SCI. 1, 133 (2020).
2. Xiaohui Song, Hong Wang, Lingfeng Wang. FPGA Implementation of a Support Vector Machine Based Classification System and Its Potential Application in Smart Grid. 11th International Conference on Information Technology: New Generations. 2014
3. Amezzane, I., Fakhri, Y., El Aroussi, M., Bakhouya, M. (2020). Hardware Acceleration of SVM Training for Real-Time Embedded Systems: Overview. In: Dos Santos, S., Maslouhi, M., Okoudjou, K. (eds) Recent Advances in Mathematics and Technology. Applied and Numerical Harmonic Analysis. Birkhäuser, Cham.
4. Frank Bruno, Guy Eschemann. The FPGA Programming Handbook: An Essential Guide to FPGA Design for Transforming Your Ideas into Hardware Using SystemVerilog and VHDL, Second Edition. 2024. 392.
5. Daniel H. Noronha, Matheus F. Torquato, Marcelo A.C. Fernandes. A parallel implementation of sequential minimal optimization on FPGA, Microprocessors and Microsystems, Volume 69, 2019, Pages 138-151.
6. Shabarinath B B, Muralidhar Pullakandam, SoC-based real-time SVM classification with integrated training using HLS and PYNQ, Microprocessors and Microsystems, Volume 101, 202.
7. Sunday O. Oladejo, Stephen O. Ekwe, Adedotun T. Ajibare, Lateef A. Akinyemi, Seyedali Mirjalili, Chapter 36 - Tuning SVMs' hyperparameters using the whale optimization algorithm, Editor(s): Seyedali Mirjalili, Handbook of Whale Optimization Algorithm, Academic Press, 2024.
8. Lucas Amilton Martins, Felipe Viel, Laio Oriel Seman, Eduardo Augusto Bezerra, Cesar Albenes Zeferino, A real-time SVM-based hardware accelerator for hyperspectral images classification in FPGA, Microprocessors and Microsystems, Volume 104, 2024.
9. Xiaowu Dong, Yujie Liu, Jingying Yan, Chaoyi Jiang, Jing Chen, Tao Liu, Yongzhou Hu, Identification of SVM-based classification model, synthesis and evaluation of prenylated flavonoids as vasorelaxant agents, Bioorganic & Medicinal Chemistry, Volume 16, Issue 17, 2008.
10. Gizen Mutlu, Çiğdem İnan Acı, SVM-SMO-SGD: A hybrid-parallel support vector machine algorithm using sequential minimal optimization with stochastic gradient descent, Parallel Computing, Volume 113, 2022.
11. Peng Wang, Ji Guo, Lin-Feng Li, Machine learning model based on non-convex penalized huberized-SVM, Journal of Electronic Science and Technology, Volume 22, Issue 1, 2024.
12. Xiaoling Shi, Dunlan Song, Jiaming Zhang, Emad Mahrous Awwad, Nadia Sarhan, Nonlinear dynamic analysis of aircraft CFRP sandwich wings under explosive blast loading: Introducing SVM-DNN algorithm to predict dynamical information, Aerospace Science and Technology, Volume 150, 2024.
13. R. Kastner, J. Matai, S. Neuendorffer, Parallel programming for FPGAs 2018.
14. Mingji Zhang, Wael A. Mahdi, Development of SVM-based machine learning model for estimating lornoxicam solubility in supercritical solvent, Case Studies in Thermal Engineering, Volume 49,2024.

ДОДАТОК Б. Презентація
(обов'язковий)




МЕТА І ЗАДАЧІ ДОСЛІДЖЕННЯ
1
розглянути застосування та реалізації FPGA
2
дослідити відомі класифікатори машинного навчання
3
проаналізувати існуючі методи реалізації класифікаторів
4
розробити оптимальний метод створення SVM-класифікатора для FPGA
5
оцінити ефективність розробленого методу та порівняти з існуючими підходами


НАУКОВА НОВИЗНА ТА ПРАКТИЧНА ЦІННІСТЬ ОТРИМАНИХ РЕЗУЛЬТАТІВ
1
Запропоновано удосконалений метод створення SVM-класифікатора для аналізу даних, який на відміну від відомих здійснює аналіз даних за допомогою застосування апаратних можливостей FPGA, що забезпечує зменшення часу аналізу даних, підвищення точності, зменшення програмно-апаратних ресурсів.
2
Розроблено програмно технічний засіб інтеграції SVM-класифікатора на базі FPGA для аналізу даних, який полягає в тому, що використовується апаратна реалізація математичних операцій для швидкого обчислення.

АКТУАЛЬНІСТЬ ДОСЛІДЖЕННЯ

1. Актуальність роботи полягає в розробці методу створення SVM-класифікатора для аналізу даних на базі FPGA, що відомі своєю високою швидкодією та можливістю паралельної обробки даних, що робить їх ідеальним кандидатом для ефективної реалізації класифікаторів машинного навчання.

2. Дослідження даної теми є важливим з огляду на потребу у швидкодії та енергоефективності, що є критичними в сучасних обчислювальних системах. А також у широкому спектрі застосувань методів класифікації в різних сферах, а саме в медичній діагностиці, фінансовому аналізі, обробці сигналів та багатьох інших областях, де вимагається аналіз великих обсягів даних у реальному часі.

МЕТОД СТВОРЕННЯ SVM-КЛАСИФІКАТОРА ДЛЯ АНАЛІЗУ ДАНИХ НА БАЗІ FPGA.

- SVM (Support Vector Machine) - це потужний і ефективний алгоритм, який може опрацьовувати набори даних навіть з великою кількістю ознак.
- Головна мета SVM - знайти оптимальну розділюючу гіперплощину, яка максимізує відстань між двома класами даних.
- SVM може ефективно використовуватися для розділення нелінійних класів, завдяки концепції ядерних функцій.
- Модель SVM оптимізується шляхом максимізації проміжку між двома класами.
- Переваги SVM включають високу точність класифікації, ефективність в просторах великої вимірності та здатність працювати зі складними нелінійними відображеннями за допомогою ядр.
- FPGA може ефективно виконувати паралельні обчислення, що дозволяє використовувати потужність SVM для швидкої класифікації в реальному часі.

МЕТОД СТВОРЕННЯ SVM-КЛАСИФІКАТОРА ДЛЯ АНАЛІЗУ ДАНИХ НА БАЗІ FPGA.

- Точність та процес класифікації SVM можуть бути в подальшому вдосконалені, використовуючи нелінійні методи оптимізації для пошуку оптимальних рішень. Тому було запропоновано інтегрувати метод оптимізації опуклої функції (один з найкращих нелінійних методів оптимізації) до класифікатора SVM, щоб чітко відрізняти два окремих класи шляхом максимізації ширини маржі гіперплощини, що в свою чергу призводить до оптимального рішення
- Одним із найбільш відомих прикладів використання опуклої оптимізації у машинному навчанні є метод опорних векторів (SVM), який використовується для класифікації та регресії. У цьому методі, завдання класифікації редукується до пошуку оптимальної гіперплощини, яка найкращим чином розділяє дані різних класів. Цей пошук гіперплощини формулюється як задача опуклої оптимізації.
- Застосування опуклої оптимізації у машинному навчанні дозволяє знаходити оптимальні рішення на великих обсягах даних, забезпечуючи ефективність та стабільність процесу навчання.

МЕТОД СТВОРЕННЯ SVM-КЛАСИФІКАТОРА ДЛЯ АНАЛІЗУ ДАНИХ НА БАЗІ FPGA.

- Загальний приріст швидкодії оцінюється за допомогою двох різних наборів тестових даних, отриманих з репозиторію машинного навчання UCI: датасет «Prediction of Pulsar Star» (набір даних для прогнозування зірки-пульсар) та датасет «Iris Flowers» (набір даних квіти Іриси) для машинного навчання.
- Цей набір даних стосується ідентифікації зірок-пульсарів, які є рідкісним типом нейтронних зірок, що випромінюють радіохвилі, які можна виявити на Землі.
- У цьому наборі даних метою є використання алгоритмів класифікації для розрізнення двох типів вибірок: реальних пульсарів (позитивний клас) і фальшивих сигналів, викликаних перешкодами або шумом (негативний клас).
- Датасет «Iris Flowers» (Іриси Фішера) - це багатовимірний набір даних для задачі класифікації. За даними вимірами будують правила класифікації, що дозволяє визначити вид рослини.

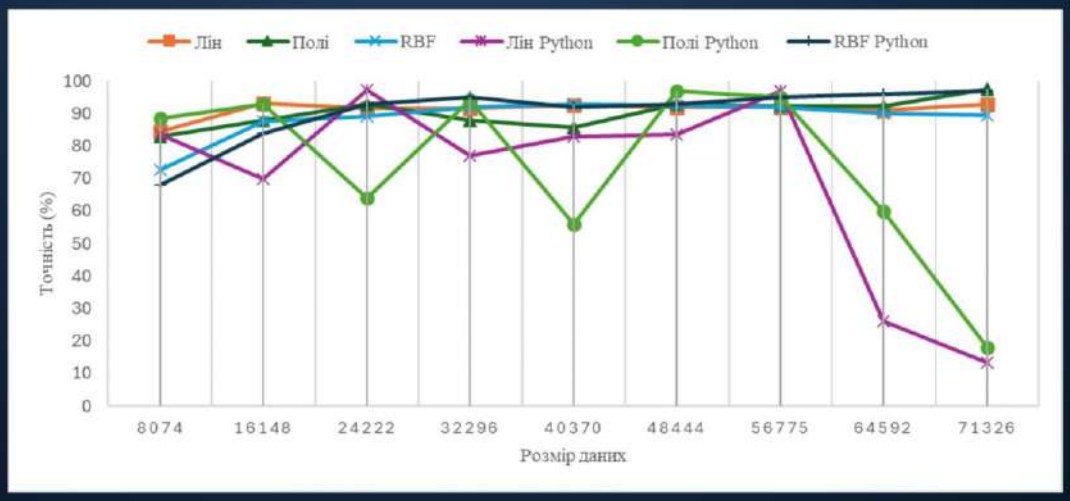
МЕТОД СТВОРЕННЯ SVM-КЛАСИФІКАТОРА ДЛЯ АНАЛІЗУ ДАНИХ НА БАЗІ FPGA.

- SVM алгоритм на основі опуклої оптимізації розбивається на три етапи, щоб спростити процес проектування. Операції цих трьох етапів - це математичні ядра, опукла оптимізація та тестування.
- Перший етап включає обчислення математичного ядра за допомогою математичних ядер (включаючи лінійні, поліноміальні та ядро Гаусса).
- Другий Етап оптимізації є найбільш складною операцією серед трьох етапів алгоритму SVM. …
- Етап 3 - це процес тестування. Зазвичай для класифікації вхідний набір даних розділяється на два зразки: навчальний і тестовий. Для процесу тестування вхідні вектори даних класифікуються у клас -1 (мінус один) або +1 (плюс один)

ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ЗАПРОПОНОВАНОГО МЕТОДУ

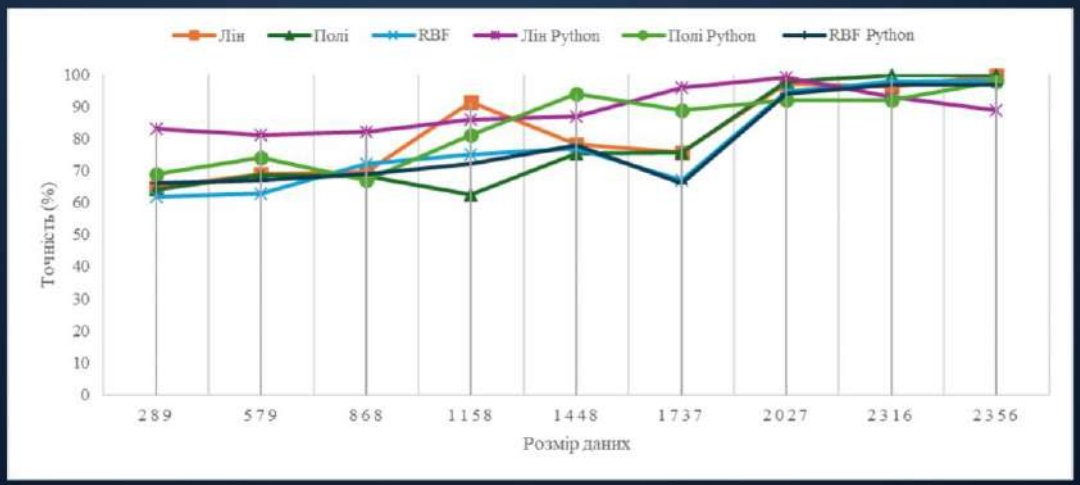
- У даному дослідженні тестування було проведено для оцінки запропонованого вбудованого апаратного та програмного забезпечення на двох різних наборах даних для тестування: набір даних «Prediction of Pulsar Star» та набір даних «Iris Flowers» для застосувань машинного навчання.
- Розмір даних вимірюється з урахуванням кількості векторів/вибірок (n) та кількості вимірів/ознак (m) у кожному векторі.
- Для всіх експериментів набори даних було розділено на два: навчальний та тестовий. Тестовий набір розглядається як відсоток набору даних для вивчення точності класифікації.
- У цьому випадку навчальний набір змінюється від 10% до 90% з кроком 10%.

АНАЛІЗ ТОЧНОСТІ КЛАСИФІКАЦІЇ



Графік точності класифікації та розміру даних для набору даних-експериментів «Pulsar»

АНАЛІЗ ТОЧНОСТІ КЛАСИФІКАЦІЇ



Графік точності класифікації та розміру даних для набору даних-експериментів «Iris Flowers»

Вбудоване програмне забезпечення для алгоритму SVM : Графік часу виконання та розміру даних для набору даних-експериментів «Pulsar»

Вбудоване апаратне забезпечення для алгоритму SVM : Графік часу виконання та розміру даних для набору даних-експериментів «Pulsar»

ПУБЛІКАЦІЇ

- За темою кваліфікаційної роботи магістра опублікована одна стаття у фаховому науковому журналі «Computer systems and information technologies» у 2024 році.

ВИСНОВКИ

В результаті досліджень було:

- розглянуто застосування та реалізації FPGA;
- досліджено відомі класифікатори машинного навчання;
- проаналізувати існуючі методи реалізації класифікаторів;
- розроблено оптимальний метод створення SVM-класифікатора для FPGA;
- здійснена оцінка ефективності запропонованого методу порівняно з існуючими підходами.

UNICHECK
by Turnitin

Ім'я користувача:
Кафедра КІ

ID перевірки:
1016247633

Дата перевірки:
13.05.2024 17:36:03 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
13.05.2024 18:44:58 EEST

ID користувача:
100005591

Назва документа: Шпуляр_Метод створення SVM-класифікатора для аналізу даних на базі FPGA

Кількість сторінок: 99 Кількість слів: 20216 Кількість символів: 153728 Розмір файлу: 3.35 MB ID файлу: 1016032797

10.1%
Схожість

Найбільша схожість: 2.98% з Інтернет-джерелом (https://link.springer.com/article/10.1007/s41870-017-0080-1)

9.81% Джерела з Інтернету | 925 ..... Сторінка 101

0.99% Джерела з Бібліотеки | 48 ..... Сторінка 108

0.64% Цитат

Цитати | 9 ..... Сторінка 109

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи | 23

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 9%**

ID: 126088 Назва: МКР Метод створення SVM-класифікатора для аналізу даних на базі FPGA Додано в БД: 2024-05-13 Автора: Шпуляр Є. М. Керівники: Говорущенко Т.О. Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	131697	965	1690 (1%)	27 (3%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувач: Шпуляр Євгенія Миколаївна

Тема: Метод створення SVM-класифікатора для аналізу даних на базі FPGA

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень – ; кількість сторінок записки 85

1. Короткий зміст роботи та прийнятих рішень У роботі запропоновано створення методу SVM-класифікатора для аналізу даних на базі FPGA.

2. Висновок про відповідність роботи дипломному завданню Кваліфікаційна робота магістра відповідає виданому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі досліджено можливості використання FPGA для аналізу даних. Було проаналізовано різні класифікатори машинного навчання, їх переваги та недоліки. У другому розділі було досліджено та проаналізовано важливі аспекти оптимальної гіперплощини, опуклої оптимізації. У третьому розділі було досліджено та проаналізовано основи створення SVM-класифікатора для аналізу даних на базі FPGA. Виявлено, що використання методу опорних векторів у поєднанні з опуклою оптимізацією дозволяє ефективно розв'язувати задачі класифікації навіть для складних та нелінійних наборів даних. У четвертому розділі запропоновано вбудований апаратний прискорювач для SVM методу на основі опуклої оптимізації та проведене експериментальне дослідження, в результаті якого було досягнуто значного покращення продуктивності та підвищення ефективності.

4. Позитивні сторони роботи: Запропонований метод є удосконаленим та оптимізованим порівняно з іншими підходами, та забезпечує значне покращення

швидкості та ефективності аналізу даних, що дозволяє досягти великої точності класифікації при значно менших часових затратах на обробку даних.

5. Негативні сторони роботи: У першому розділі приділено недостатньо увагу іншим відомим класифікаторам.

6. Оцінка графічного оформлення та пояснювальної записки роботи:

7. Відгук про роботу в цілому: В загальному робота виконана на високому рівні.

8. Інші зауваження: –

9. Оцінка кваліфікаційної роботи:

Розглянувши позитивні та негативні сторони представленої кваліфікаційної роботи вважаю, що робота заслуговує оцінки «відмінно» 4.80 (A)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) д.т.н. професор, Бармак О. В., завідувач кафедри комп'ютерних наук

"13 травня" 2024р.

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т. О.

Шпуляр Євгенії Миколаївни

ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-22-1

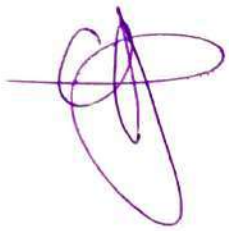
ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

22 квітня 2024 року



РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ
КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОМАЦІЙНИХ СИСТЕМ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

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

Назва: Метод створення SVM-класифікатора для аналізу даних на базі FPGA

Автор: Шпуляр Євгенія Миколаївна

Спеціальність: 123 – Комп'ютерна інженерія

Освітня програма: освітньо-наукова

Науковий керівник: Іванов О. В. к.т.н. доцент

Після аналізу звіту подібності зроблено такий висновок:

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

Підтвердження:

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

1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
2) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;
3) в якості запозичень в окремих місцях системою зафіксовано послідовності чотирьохрозряних двійкових кодів, які є вхідними даними до великої кількості задач і не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості Unicheck, складає 10.1% і адресується до 925 першоджерела; та системою Anti-Plagiarism складає 1%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ О.В. Іванов

Гарант ОП _____ О. С. Савенко

Завідувач кафедри КІІС _____ Т. О. Говорущенко