БАКАЛАВРСЬКА РОБОТА

БР.ІП – 35.00.00.000 ПЗ

Група ІП-22-2

Кузьмин Богдан

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Кузьмин Богдан Богданович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Розробка енергоефективного програмного забезпечення
(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____Б.Б._ Кузьмин _____
(підпис, ініціали та прізвище здобувача)

Науковий керівник ____Михайлюк Ірина Романівна, к.п.н., доцент_____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.__
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2026 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Кузьмин Богдану Богдановичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Розробка енергоефективного програмного забезпечення"

керівник проекту (роботи) Михайлюк Ірина Романівна, к.п.н., доцент

затверджені наказом вищого навчального закладу від “ 21 ” квітня 2026 р. № 179/7

2. Строк подання студентом проекту (роботи) 10 червня 2026 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

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

1. Аналіз вимог до програмного забезпечення
2. Аналіз вимог і постановка задачі
3. Проектування системи
4. Реалізація проекту
5. Тестування та впровадження

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1 SDLC для таксономії енергоефективності (рис. 1.1, ст. 13)

2 Проведення експерименту (рис. 2.1, ст. 25)

3 - Діаграми розсіювання змінних CPUusr і Watts до та після застосування практики 1 (рис. 3.1, ст. 58)

4 Діаграми розсіювання змінних CPUusr і Watts до та після застосування практики 2 (рис. 3.2, ст. 60)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання **2026 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обгрунтування теми роботи	17.01.2026	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	21.02.2026	виконано
3	Побудова моделі або алгоритму власного рішення	01.03.2026	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	21.04.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	02.05.2026	виконано

Студент _____Кузьмин Б.Б.
(підпис)

Керівник роботи _____Михайлюк І.Р.
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 69 сторінок, 4 рисунків, 12 таблиць, список використаних джерел із 40 найменувань.

Метою роботи є дослідження принципів розробки енергоефективного програмного забезпечення та аналіз підходів до побудови програмних систем із мінімальним енергоспоживанням при збереженні високої продуктивності.

Об'єкт дослідження: процеси розробки енергоефективного програмного забезпечення.

Предмет дослідження: методи, моделі та інструменти підвищення енергоефективності програмних систем, включаючи оптимізацію алгоритмів, архітектурні підходи та засоби вимірювання енергоспоживання.

Результати дослідження: проведено аналіз підходів до розробки енергоефективного програмного забезпечення, визначено основні вимоги та принципи енергоефективності, досліджено методи оптимізації.

У першому розділі розглянуто теоретичні основи енергоефективного програмного забезпечення, визначено ключові поняття, проблеми та вимоги до таких систем.

У другому розділі досліджено методи та інструменти розробки енергоефективних програмних систем, включаючи архітектурні підходи, оптимізацію виконання та використання паралельного програмування.

У третьому розділі розглянуто практичні аспекти реалізації енергоефективних рішень, проаналізовано інструменти вимірювання енергоспоживання.

Висновок: застосування методів енергоефективної розробки дозволяє зменшити споживання ресурсів, підвищити продуктивність програмних систем та знизити витрати на їх експлуатацію.

КЛЮЧОВІ СЛОВА: ЕНЕРГОЕФЕКТИВНІСТЬ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ОПТИМІЗАЦІЯ, ПАРАЛЕЛЬНЕ ПРОГРАМУВАННЯ, ПРОДУКТИВНІСТЬ, ЕНЕРГОСПОЖИВАННЯ, ПРОГРАМНА ІНЖЕНЕРІЯ.

ANNOTATION

The bachelor's thesis contains of 69 pages, 4 figures, 12 tables, and a reference list with 40 sources.

The aim of the work is to study the principles of energy-efficient software development and to analyze approaches to building software systems with minimal energy consumption while maintaining high performance and quality.

Research object: processes of energy-efficient software development.

Research subject: methods, models, and tools for improving the energy efficiency of software systems, including algorithm optimization, architectural approaches, and energy consumption measurement techniques.

Research results: an analysis of approaches to energy-efficient software development was conducted, key requirements and principles were identified, methods for optimizing program execution were studied, and tools for measuring and evaluating energy consumption were reviewed.

The first section describes the theoretical foundations of energy-efficient software, including key concepts, challenges, and requirements.

The second section analyzes methods and tools for developing energy-efficient systems, including architectural approaches, execution optimization, and parallel programming techniques.

The third section covers the implementation of energy-efficient solutions, energy consumption measurement tools, software maintenance aspects.

Conclusion: the use of energy-efficient development methods reduces resource consumption, improves system performance, and lowers operational costs, which is especially important in modern IT environments and increasing demands for environmentally sustainable technologies.

KEY WORDS: ENERGY EFFICIENCY, SOFTWARE, OPTIMIZATION, PARALLEL PROGRAMMING, PERFORMANCE, ENERGY CONSUMPTION, SOFTWARE ENGINEERING.

ЗМІСТ

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій та зростання обсягів обчислень питання енергоефективності програмного забезпечення набуває особливої актуальності. Збільшення кількості дата-центрів, мобільних пристроїв і розподілених систем призводить до значного зростання споживання електроенергії, що, у свою чергу, впливає на економічні витрати та екологічний стан довкілля. У цьому контексті розробка енергоефективного програмного забезпечення стає важливим напрямом програмної інженерії, спрямованим на зменшення енергоспоживання без втрати продуктивності та функціональності систем.

Актуальність теми зумовлена необхідністю створення програмних рішень, які б оптимально використовували обчислювальні ресурси, зменшували навантаження на апаратне забезпечення та сприяли зниженню витрат енергії. Існуючі програмні системи часто орієнтовані переважно на продуктивність і функціональність, і лише частково враховують аспекти енергоефективності. У зв’язку з цим виникає потреба у дослідженні сучасних підходів, методів і інструментів, які дозволяють забезпечити баланс між продуктивністю, якістю програмного забезпечення та його енергоспоживанням.

Метою роботи є дослідження принципів та методів розробки енергоефективного програмного забезпечення, а також аналіз підходів до створення програмних систем із мінімальним рівнем енергоспоживання.

Завданнями дослідження є: аналіз теоретичних основ енергоефективності програмного забезпечення; визначення основних вимог та проблем у даній сфері; дослідження методів і підходів до оптимізації енергоспоживання програм; аналіз сучасних інструментів вимірювання та оцінки енергоефективності; розробка та оцінка ефективності програмного рішення з урахуванням енергоефективних підходів.

Об’єктом дослідження є процеси розробки програмного забезпечення з урахуванням вимог енергоефективності.

Предметом дослідження є методи, моделі та інструменти забезпечення енергоефективності програмних систем, зокрема архітектурні підходи, алгоритмічна оптимізація та засоби контролю енергоспоживання.

Методи дослідження включають аналіз наукових джерел для вивчення теоретичних основ, порівняльний аналіз існуючих підходів і технологій, моделювання та оптимізацію програмних рішень, а також експериментальні методи для оцінки енергоспоживання та продуктивності систем.

У роботі передбачається дослідження принципів проектування енергоефективних систем, використання сучасних методів оптимізації виконання програм, а також аналіз інструментів для вимірювання енергоспоживання. Особлива увага приділяється питанням підвищення продуктивності, зменшення використання ресурсів та забезпечення стабільної роботи програмних систем. Очікується, що результати дослідження сприятимуть підвищенню ефективності програмних продуктів та зниженню їх впливу на навколишнє середовище.

Бакалаврська робота містить 69 сторінок, 4 рисунки, 12 таблиць, 3 розділи, список використаних джерел із 40 найменувань.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ЕНЕРГОЕФЕКТИВНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Поняття та принципи енергоефективності програмного забезпечення

Вплив програмного забезпечення на енергоспоживання було визнано значним щодо загального споживання енергії його середовищем виконання [1, 2]. Багато дослідників працюють над складними моделями енергоспоживання програмного забезпечення, здатними оцінювати та прогнозувати споживання енергії програмними додатками за різними параметрами. Незважаючи на ці зусилля, для практиків та розробників немає інформації, яку можна було б повторно використовувати для створення енергоефективних програмних додатків. Крок у цьому напрямку було зроблено з Intel Corp., які надали низку рекомендацій та найкращих практик для створення енергоефективного програмного забезпечення [1]. Однак, ці практики були практично не підтверджені, а їхня ефективність з точки зору споживання енергії не була точно кількісно визначена.

Щоб зрозуміти, як програмне забезпечення може впливати на споживання енергії, розглянемо наступний приклад: після запуску популярне відео на YouTube з піснею «Gangnam Style» досягло рекордної кількості візуалізацій протягом першого року після публікації - приблизно 1,7 мільярда. Кількість енергії, яку Google використовує для передачі 1 МБ через Інтернет (як повідомляє компанія на своєму веб-сайті), становить 0,01 кВт·год (приблизне середнє значення), і для його відображення використовується 002 кВт·год (залежно від пристрою призначення). Отже, енергія, необхідна для потокової передачі та відображення відео «Gangnam Style», становить 0,19 кВт·год. Помноживши цю кількість енергії на 1,7 мільярда візуалізацій, отримаємо 312 ГВт·год загального споживання енергії, що приблизно дорівнює річній потребі в енергії міста з населенням 22 000 мешканців (наприклад, місто Ізернія, Італія, спожило 340 ГВт·год електроенергії у 2013 році [6]).

Однак, ця вражаюча кількість енергії може приховувати величезні втрати.

Складна архітектура програмного забезпечення лежить в основі сучасних веб-додатків та сервісів (наприклад, веб-серверів, серверів баз даних, проміжного програмного забезпечення), і незліченна кількість екземплярів виконується щосекунди у фізичному та віртуальному середовищах. Навіть невелика оптимізація одного програмного застосунку в такому великому масштабі потенційно може призвести до значної економії енергії. З цієї причини архітекторам та розробникам програмного забезпечення необхідно думати про енергоефективність, і необхідна міцна база знань, щоб надати рекомендації щодо створення енергоефективного програмного забезпечення.

Метою цієї роботи є оцінка впливу двох найкращих практик енергоефективної розробки програмного забезпечення на споживання енергії. Ми хочемо відповісти на такі дослідницькі питання: 1. який вплив кожної практики на енергоспоживання, які основні фактори, що спричиняють такий вплив (тобто системні ресурси).

Наша робота відповідає рекомендаціям щодо емпіричних експериментів у програмній інженерії. Для цілей цього експерименту ми обрали дві найкращі практики енергоефективної розробки програмного забезпечення. Ми виявили ці практики, натхненні академічною літературою та промисловістю, та зібрали їх у вікі. щоб поділитися ними з науковцями та практиками. Такі методи були обрані, оскільки вони застосовні до широко використовуваних та відомих програмних застосунків з відкритим кодом (зокрема, ми обрали Apache WebServer та MySQL Database Server), які слугуватимуть тестовими випадками. Ці застосунки виконувалися в контрольованому середовищі (Лабораторія енергетичного сліду програмного забезпечення, SEFLab). Під час експерименту ми зібрали два типи даних: споживання енергії (як середовища виконання в цілому, так і окремих апаратних компонентів) та коефіцієнт використання різних системних ресурсів [2]. Потім ми провели перевірку гіпотез даних, щоб відповісти на наші дослідницькі питання. Окрім оцінки впливу кожної практики на енергію, ми виявили для кожної реалізації тестового випадку фактори, які ми визначили як найбільш релевантні для цілей споживання енергії. З цього порівняння ми витягли значущу інформацію для

подальшого визначення складного взаємозв'язку між програмним забезпеченням та енергією.

Цей внесок належить до сфери дослідження «зеленого» програмного забезпечення. «Зелене» програмне забезпечення можна охарактеризувати двома способами: (1) програмний код є «зеленим», тобто має знижений вплив на навколишнє середовище, зазвичай з точки зору споживання енергії, незалежно від його призначення (як у «зелених» ІТ або «озелененні» ІТ), або (2) призначення програмного забезпечення є «зеленим», що покращує вплив на навколишнє середовище підтримуваних процесів у контексті їх використання (також позначається як «озеленення» ІТ [16]). Зокрема, наш внесок зосереджений на енергоефективності програмних застосунків, тому ми називаємо «зелене» програмне забезпечення, як у (1). З цієї причини в решті цієї роботи терміни «зелене» програмне забезпечення та «енергоефективне програмне забезпечення» будуть використовуватися як взаємозамінні.

Сучасні технології (наприклад, розподілені системи, мобільні обчислення, віртуалізація та хмарні обчислення) ускладнюють взаємозв'язок між апаратним та програмним забезпеченням, особливо з точки зору споживання енергії [3]. З цієї причини ми розглядаємо енергоефективність як емерджентну властивість програмних систем, що використовуються: складність взаємодії апаратно-програмного забезпечення та численні програмні рівні створюють середовище, яке ми наразі не можемо детерміновано описати. Отже, наше дослідження використовує індуктивний підхід, тобто ми накопичуємо знання про енергоефективність програмного забезпечення шляхом збору та аналізу емпіричних даних [17].

Такі докази, очевидно, потребують контекстуалізації, щоб забезпечити узгоджений корпус знань. Зокрема, апаратне середовище виконання відіграє дуже важливу роль. Наприклад, як ми покажемо в нашому аналізі пов'язаної роботи в наступному розділі, проведення експериментів на мобільних пристроях, серверах або вбудованих системах призводить до різноманітних висновків, які часто можуть бути суперечливими. Саме тому, окрім простої оцінки впливу найкращих практик,

у нашому дослідженні ми також намагаємося пояснити механізми такого впливу, вимірюючи та аналізуючи контекстно-специфічні дані [4]. У цій роботі ми обираємо невелику кількість найкращих практик та застосувань як предмети дослідження – рішення, яке явно ставить під сумнів узагальнення наших результатів, – але ми зосереджуємося на точності вимірювань та методах ретельного аналізу даних з метою підвищення внутрішньої валідності наших експериментів.

Оскільки «зелене» програмне забезпечення стало популярною темою дослідження, було проведено низку емпіричних експериментів щодо енергоспоживання програмного забезпечення [5]. У цьому розділі ми представляємо ті з них, які більше пов’язані з нашим внеском, упорядковані за датою публікації, та підсумовуємо їхні результати [6]. У таблиці 1 ми наводимо більш структурований огляд: ми перераховуємо мету дослідження , експериментальний контекст (наприклад, онлайн проти офлайн [7]), вибрані для дослідження суб’єкти та тестовий стенд , на якому проводилися вимірювання енергоспоживання (де це можливо). Як критерії відбору ми зосередилися на точку зору розробників: отже, ми вибрали дослідження, що аналізують вплив методів або практик програмування на енергоспоживання, а також дослідження, які намагаються емпірично охарактеризувати енергоємні елементи коду.

У цій роботі ми обмежуємо сферу нашого інтересу існуючими роботами, пов'язаними з енергоефективністю, зосередженими на кожному етапі SDLC, тобто Вимоги, Проектування, Впровадження, Верифікація та Обслуговування, дотримуючись каскадної моделі. Хоча каскадна модель SDLC є застарілим підходом до розробки програмного забезпечення, вона все ще корисна як еталонна модель для категоризації досліджень у цій галузі. На рисунку 1.1 ми представляємо схему методів та інструментів енергоефективності в рамках процесу SDLC . Дотримуючись цієї класифікації, ми структуруємо цю роботу відповідно, щоб надати повне уявлення про існуючі інструменти та методи для енергосвідомої розробки програмного забезпечення.

Фаза вимог описує аспекти та потреби, що стосуються розробки програмних

проектів/систем, коли енергоефективність є обов'язковою. На фазі проектування шаблони проектування програмного забезпечення документують найкращі практики, що використовуються для вирішення спільних проблем або поганих проектних рішень, прийнятих під час розробки програми, які можуть вплинути на споживання енергії.

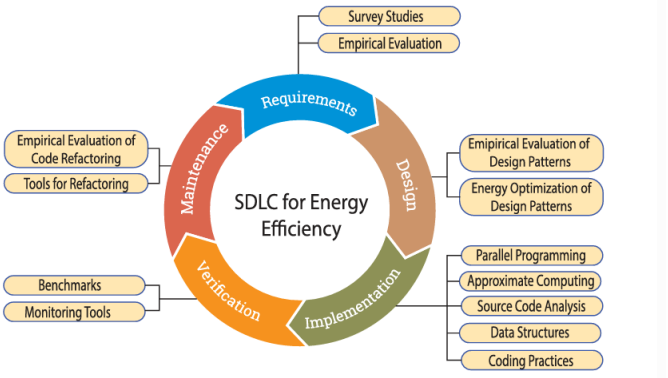

Рисунок 1.1 - SDLC для таксономії енергоефективності

Фаза впровадження об'єднує методи/практики, а саме: паралельне програмування, наближені обчислення, аналіз вихідного коду, мови програмування, структури даних та практики кодування, які фахівці можуть застосовувати для зменшення споживання енергії програмним забезпеченням під час розробки [7]. Фаза перевірки розглядає метрики та інструментарій підтримки з метою оцінки споживання енергії програмним забезпеченням перед його розгортанням. Підтримка – це фаза, метою якої є застосування шаблонів/методів рефакторингу до розгорнутої програми для зменшення її споживання енергії.

1.2 Актуальні проблеми та дослідницькі питання у сфері енергоефективності

У цьому розділі ми описуємо мету нашого експериментального дослідження, натхненного підходом . Як описано, метою цієї роботи є оцінка впливу низки екологічно чистих Практика розробки програмного забезпечення з точки зору споживання енергії, з точки зору розробників додатків, у контексті

програм з відкритим кодом.

ЗП1: Який вплив кожної практики має на споживання енергії?

Ми відповідаємо на це питання, застосовуючи практики «зеленого» програмного забезпечення до вже існуючих програмних застосунків, а потім аналізуючи значення споживання енергії, спричинені виконанням застосунків до та після впровадження практики. Оцінка впливу кожної практики допомагає розробникам визначити їх пріоритетність відповідно до очікуваних переваг. Вплив (ДЕ) вимірюється у ват-годинах (Вт·год) та виражає різницю між споживанням енергії на рівні системи до (E0) та після (Ei) застосування практики.

ЗП2: Чи впливає застосування кожної практики на співвідношення між ресурсами та споживанням енергії?

Це запитання зосереджено на використанні ресурсів: кожна програмна програма має різну схему використання системних ресурсів (процесор, оперативна пам'ять, жорсткі диски тощо). Як показано в попередніх експериментах [2], існує відповідна кореляція між використанням ресурсів та споживанням енергії. Відповідь на це запитання дозволяє визначити, які типи ресурсів впливають на споживання енергії, і як застосування певної практики може змінювати схему використання. Це допомагає розробникам контролювати найбільш доцільні ресурси, визначати їх пріоритети та встановлювати компроміси між нефункціональними аспектами (наприклад, енергоефективність та продуктивність). Крім того, це дозволяє визначити, чи має програма енергопропорційну поведінку: в ідеально енергопропорційній системі споживання енергії є лінійною функцією використання ресурсів [34]. Досягнення такої умови дозволяє визначити постійну кількість енергії на певне робоче навантаження, отже, це дає точний показник енергоефективності. Наразі це є проблемою для IT-систем, значною мірою через технологічні обмеження; отже, метод програмного рівня, який підвищує енергопропорційність, був би дуже корисним.

На це питання відповідає аналіз індексу кореляції між споживанням енергії та статистикою використання ресурсів до та після застосування практики [8]. У цьому випадку використовується потужність (а не енергія), оскільки нас цікавить

аналіз динамічної поведінки ресурсів. Зокрема, ми визначаємо u/ $_r$ як коефіцієнт використання певного ресурсу r, P як споживання енергії на системному рівні та Pr як споживання енергії компонентом, відповідальним за надання певного ресурсу (наприклад, процесором для обчислювальних ресурсів, банками пам'яті для оперативної пам'яті, материнською платою для інших периферійних пристроїв вводу/виводу тощо). Ми також визначаємо :

- cor(u r ,P) як індекс кореляції між u r та P перед застосуванням практики
- cor(u! r , P') як індекс кореляції між u! r та P після застосування практики
- cor(ur , Pr) як індекс кореляції між u r та Pr перед застосуванням практики
- cor(u! r , Pr) як індекс кореляції між u !r та Pr після застосування практики

Для нашого аналізу ми скористаємося коефіцієнтом рангової кореляції Спірмена [35]: можливі значення цього індексу включені в інтервал [-1,1], де кінцеві точки інтервалу вказують на ідеально лінійну функцію між двома змінними (отже, у нашому випадку, ідеально степенево-пропорційну поведінку ресурсу).

Основним об'єктом нашого дослідження є вплив на енергію двох найкращих практик енергоефективності програмного забезпечення, або практик «зеленого» програмного забезпечення. Ми виявили ці практики, натхненні академічною літературою та галуззю та зібрали їх у вікі, щоб поділитися ними з науковцями та практиками.

Для цієї оцінки ми обрали дві практики з нашої вікі: «Використовувати ефективні запити» та «Переводити програму в режим сну». Ці практики були обрані з двох основних причин: висока актуальність для практиків, оскільки їх можна застосовувати в широкому контексті програмних застосунків, та їх чітко визначена сфера впровадження, що дозволяє краще відстежувати їхній вплив.

Через природу практик та їх формалізації, не було можливості автоматизувати та рандомізувати їх застосування до досліджуваних суб'єктів, тобто програмних за стосунків [9]. Отже, наше емпіричне дослідження кваліфікується як квазіексперимент**,** оскільки методи лікування застосовувалися вручну до двох

вибраних програмних застосунків. Ми обрали два поширені продукти з відкритим кодом: веб-сервер Apache та сервер баз даних MySQL. Цей вибір був зумовлений тими ж критеріями, що й вибір об'єктів: широке використання цих продуктів забезпечує їхню актуальність для практиків, а їхня відкритість дозволила нам легко отримати доступ до їхнього вихідного коду для цілей інструментального дослідження.

Залежними змінними**,** які ми контролювали та аналізували для відповіді на наші запити, є: споживання енергії на системному рівні для оцінки впливу практик на енергію; значення споживання енергії на рівні ресурсів та їх коефіцієнт використання для визначення ресурсів, на які найбільше впливає зміна; та показники виконання програмного забезпечення (час відгуку, кількість оброблених запитів) для визначення їхнього зв'язку зі споживанням енергії та того, як на них впливає застосування практики.

Основною незалежною змінною для нашого експерименту є застосування практик, яке ми обрали як наш основний фактор. Цей фактор визначає два підходи: чи застосовується практика «зеленого» програмного забезпечення до досліджуваного програмного застосунку, чи ні (відсутність підходів). Ми визначили та контролювали інші незалежні змінні, щоб уникнути потенційного змішування [10].

1.3 Вимоги до енергоефективного програмного забезпечення

Класифікація енергоефективності для SDLC належить до нефункціональних вимог, таких як продуктивність під час виконання та безпека [11]. Ми представляємо відповідну роботу щодо вимог, зосереджену на пропозиціях, зібраних фахівцями-опитувальниками, та на результатах, заснованих на емпіричній оцінці. Відповідно, ми поділяємо розділ « Вимоги» на два підрозділи: дослідження опитувань та вимоги до емпіричної оцінки.

Мобільні пристрої більш енергозалежні, ніж сервери чи робочі станції; тому вкрай важливо застосовувати енергосвідомий підхід та враховувати термін служби

батареї як обмеження перед розробкою мобільного додатку. Якісне дослідження, розглядало перспективи екологічної, енергоефективної розробки програмного забезпечення 464 фахівців з відділів розробки програмного забезпечення ABB, Google, IBM та Microsoft. Хоча дослідження стосувалося різних обчислювальних систем, зокрема мобільних та вбудованих систем і центрів обробки даних, фахівці висловлювали ідеї, здебільшого, щодо того, як зменшити споживання енергії в мобільних додатках. Деякі фахівці також навели приклади щодо сталого споживання енергії смартфонами, такі як «покрокова навігація не повинна розряджати більше батареї, ніж може зарядити автомобіль» та «за нормального використання пристрій з батареєю $a\,n\,X\ \ В\,т\cdot г\,о\,д$ повинен працювати протягом Y годин» [12]. Крім того, фахівці пропонують виконувати певні завдання, не турбуючи користувачів про розрядку батареї.

Провели дослідження, в якому опитали 122 програмістів онлайн, і, подібно до Манотаса та ін., дійшли висновку, що фахівці з програмного забезпечення здебільшого розглядають споживання енергії як вимогу для розробки мобільних додатків. З цією метою автори стверджували, що розробники можуть вивести вимоги до енергоефективності, співвідносячи функціональні вимоги додатків зі споживанням енергії відповідними компонентами програмного забезпечення. Для досягнення цієї мети необхідний узгоджений обсяг знань та розуміння взаємодії програмного та апаратного забезпечення [13]. Автори перерахували наступні відповідні інструкції, які розробники можуть враховувати:

- Масові операції, щоб мінімізувати кількість викликів вводу/виводу.
- Координація обладнання, така як мінімізація доступу до пам'яті.
- Паралельне програмування, таке як вибір відповідної конструкції потоку.
- Ефективні структури даних, вибір менш енергоємних структур даних.
- Перетворення циклів, таке як об'єднання циклів для зменшення кількості операцій керування.
- Стиснення даних, щоб зменшити розмір файлів перед їх передачею.

- Методи розвантаження шляхом віддаленого обчислення важких операцій, тобто хмарних технологій.
- Наближене програмування для зменшення непотрібної точності обчислень.

Підхід до визначення нефункціональних вимог для зменшення споживання енергії під час розробки програмного забезпечення. Автори зосередили своє дослідження на настільних комп'ютерах та смартфонах, де вони стверджували, що для досягнення енергоефективної розробки програмного забезпечення необхідно визначити характеристики та вимоги до програмного забезпечення. Відповідно до цього, практик може враховувати чотири характеристики для забезпечення енергоефективних вимог для різних типів систем, а саме:

- Обчислення для оптимізації споживання енергії за рахунок менш витратних обчислень (залежно від процесора).
- Управління даними, для зменшення кількості операцій вводу/виводу, оскільки вони є повільними та дорогими для системи (обмежені обсягом сховища).
- Передача даних, тобто обсяг даних, що надсилаються або отримуються через мережевий канал (мережеві операції).
- Усвідомлення споживання енергії, щоб надавати інформацію, пов'язану з енергоспоживанням, для окремих рівнів програмного забезпечення (наприклад, за допомогою інструментів профілювання енергії) та для програмного забезпечення в цілому.

З цією метою розробили інструмент для тестування вимог до додатків шляхом оцінки споживання енергії ПРОЦЕСОРОМ, ОПЕРАТИВНОЮ ПАМ'ЯТТЮ та мережевою інтерфейсом. Емпірично оцінюють запропоновані ними вимоги. Орієнтовані на опитування та не надають оцінки запропонованих пропозицій. Свої вимоги від досвідчених та відомих компаній-розробників програмного забезпечення [14]. Загалом, бракує дослідницьких робіт щодо практичних рекомендацій для етапу вимог щодо енергоефективності розробки програмного забезпечення. Більшість робіт обмежувалися досвідом розробників. Як показано,

усі дослідники виявили деякі обмеження та проблеми на етапі Вимог , які можуть слугувати орієнтирами для майбутніх досліджень.

1.4 Висновки по розділу

У першому розділі було розглянуто теоретичні основи розробки енергоефективного програмного забезпечення. Проаналізовано поняття та базові принципи енергоефективності у контексті сучасної програмної інженерії. Визначено актуальні проблеми та ключові дослідницькі питання, пов’язані зі зниженням енергоспоживання програмних систем. Також сформульовано основні вимоги до енергоефективного програмного забезпечення, що включають оптимальне використання ресурсів, продуктивність та стабільність роботи. Отримані результати створюють теоретичну основу для подальшого дослідження методів і практичної реалізації енергоефективних рішень.

РОЗДІЛ 2. МЕТОДИ ТА ІНСТРУМЕНТИ РОЗРОБКИ ЕНЕРГОЕФЕКТИВНИХ СИСТЕМ

2.1 Архітектурні та дизайнерські підходи до енергоефективності

Правильні дизайнерські рішення мають вирішальне значення, коли йдеться про енергоефективну розробку програмного забезпечення, оскільки компоненти програмного забезпечення та їх взаємодія можуть суттєво впливати на споживання енергії [15]. У наступних підрозділах ми представляємо дослідження, які оцінюють енергетичні наслідки шаблонів проектування або рекомендують коригування для їх оптимізації з точки зору споживання енергії. Ми також вказуємо на випадки, коли неправильні дизайнерські рішення призводять до збільшення споживання енергії.

Шаблони проектування – це загальні та багаторазові рішення для поширених проблем проектування програмного забезпечення. У контексті шаблонів проектування, провели емпіричні дослідження, в яких порівнювали споживання енергії вибраними шаблонами. В обох роботах шаблони проектування оцінювалися з категорій створення, структури та поведінки.

Досліджували споживання енергії різними застосуваннями працює на вбудованій системі, із застосуванням шаблонів проектування та без нього. Автори проілюстрували метод кореляції проектування програмного забезпечення та споживання енергії, який допомагає розробникам програмного забезпечення зрозуміти компроміси між їхніми дизайнерськими рішеннями та споживанням енергії. Для свого експерименту вони використали 15 із 23 шаблонів проектування. Як результат, 3 з 15 використаних шаблонів призвели до суттєвої економії енергії, тоді як решта призвели до незначних змін або негативно вплинули на споживання енергії. Зокрема, шаблони Flyweight, Mediator та Proxy призвели до економії енергії при застосуванні до вибраних програм, тоді як шаблон Decorator значно збільшив споживання енергії.

Таблиця 2.1

Емпірична оцінка впливу архітектурних шаблонів на енергоефективність

Тип дослідження	Шаблон	Енергія (сер. у %)	Продуктивність (сер. у %)	Тип платформи
Емпірична оцінка	Flyweight	58	—	Вбудована система
	Proxy	36	—	
	Mediator	9.56	—	
	Composite	− 5.14	—	
	Abstract Factory	− 21.55	—	
	Observer	− 62.20	—	
	Decorator	− 712.89	—	
Емпірична оцінка	Decorator	− 133.60	− 132.40	Смартфон
	Prototype	− 33.20	− 33	
	Abstract Factory	− 14.20	− 14.20	
Оптимізація шаблонів	Observer	4.32	—	Робоча станція
	Decorator	25.47	—	

Зосереджено увагу на оцінці споживання енергії та впливу шаблонів проєктування на продуктивність під час виконання в додатках Android. Автори спостерігали збільшення як споживання енергії, так і часу виконання після застосування 6 з 23 шаблонів проєктування (Фасад, Абстрактна фабрика, Спостерігач, Декоратор, Прототип та Метод шаблонів) до вибраних додатків.

Результати, зведені в Таблиці 2.1, показують, що навіть між випадками застосування однакових шаблонів проектування існують значні відмінності у споживанні енергії. Наприклад, згідно з результатами Банса та ін., шаблон Декоратор збільшив споживання енергії на 133%, тоді як Сахін та ін. виявили значно вище споживання енергії, тобто на 712%; це може статися тому, що Банс та ін. використовували смартфон та фрагменти коду на основі Java для експериментів, тоді як Сахін та ін. використовували вбудовану систему та фрагменти коду, написані на C++.

На завершення, негативний вплив деяких шаблонів проектування у вбудованих пристроях та смартфонах щодо споживання енергії. Більше того, Банс та ін. показали, що певні шаблони проектування, що спричиняють високе споживання енергії, також сприяють зниженню продуктивності. Як зазначено в

обох дослідженнях, кількість об'єктів та комунікацій між програмними компонентами є основними факторами збільшення споживання енергії. В обох дослідженнях шаблони проектування Decorator та Abstract Factory визначені як найбільш енергоефективні шаблони проектування. Додаткова робота з використанням конкретних бенчмарків може допомогти глибше зрозуміти вплив шаблонів проектування в різних додатках та платформах.

Структурні зміни в шаблонах проектування можуть призвести до значної оптимізації енергоспоживання. Спочатку автори провели експериментальне дослідження, в якому вони вручну застосували 21 різний шаблон проектування до 11 програм, написаних як на C++, так і на Java. Їхній експеримент показав, що Observer та Decorator є найбільш енергоємними шаблонами проектування. Автори трансформували існуючий вихідний код, зменшивши кількість створених об'єктів та викликів функцій, і досягли значної економії енергії. Зокрема, після оптимізації шаблонів проектування Observer та Decorator автори зменшили споживання енергії програмами в середньому на 10%.

У роботах бракує вимірювань продуктивності під час виконання. Отже, невідомо, як оптимізація шаблонів проєктування вплинула на час виконання програм [16]. Однак така оптимізація шаблонів проєктування здається перспективною галуззю для подальших досліджень. З цією метою інструмент, який вказує на структурні зміни шаблонів проєктування, що призводять до оптимізованого споживання енергії програмами, може бути корисним для розробників програмного забезпечення.

2.2 Методи оптимізації виконання програм

Контекст нашого експерименту – це дослідження одного об'єкта : ми застосовуємо один об'єкт (тобто програмну практику) до одного суб'єкта (тобто програмного застосунку). Цей вибір був зроблений через внутрішню складність практичного застосування: наразі енергоефективні програмні практики описуються в літературі як рекомендації високого рівня, тому немає формальних

специфікацій щодо того, як застосовувати практику до застосунку. Отже, застосування однієї й тієї ж практики до кількох застосунків є складним завданням з точки зору узагальнення результатів [17]. Крім того, для нашого дослідження ми вирішили модифікувати оригінальні застосунки неінвазивним способом, за допомогою простого рефакторингу або операцій модифікації запитів. Фактично, неінвазивність є одним із критеріїв, які ми використовували для вибору наших практик. Це обмежує можливі фактори, що впливають на результати, і не ставить під загрозу оригінальну архітектуру наших застосунків, які досліджуються. У цьому розділі ми описуємо, як ми реалізували ці практики та які припущення ми зробили.

- Для кожної практики ми розробили два різні сценарії:
- Перший сценарій включає тестовий застосунок з інструментарієм коду перед застосуванням тестованої практики програмного забезпечення.
- Другий сценарій включає тестовий застосунок з інструментарієм коду після застосування тестованої практики програмного забезпечення.

Ми реалізували цю практику за допомогою програмного забезпечення MySQL Database Server [18]. Як набір даних ми використовували повну копію статей англійської Вікіпедії станом на 2008 рік, розмір якої приблизно 30 ГБ. Набір даних англійської Вікіпедії був отриманий від WikiMedia Foundation. Ми розробили простий запит, який ітерує по всіх сторінках Вікіпедії в пошуках фрагментів тексту. Ми вимкнули внутрішній кеш MySQL, який потенційно міг би бути фактором, що спотворює результат, використовуючи ключове слово SQL_NO_CACHE в SQL-інструкції. Застосування цієї практики моделюється шляхом виконання двох різних типів запитів: один використовує ключове слово ORDER BY для впорядкування результатів, інший - ні. Ми розробили бенчмарк, який виконує SQL-запит 3 рази. Ми вирішили не контролювати тривалість бенчмарку для цієї практики: вона змінюється залежно від часу виконання запитів. Це дозволяє нам оцінити вплив практики на продуктивність. У лістингах 2.1 та 2.2 показано SQL-інструкції, які ми використовували.

Лістинг 2.1 - Запит до застосування цієї методики

```
SELECT SQL_NO_CACHE a.old_id
FROM text a, revision b
WHERE a.old_id = b.rev_text_id
ORDER BY a.old_id;
```

Лістинг 2.2 - Запит після застосування цієї методики

```
SELECT SQL_NO_CACHE a.old_id
FROM text a, revision b
WHERE a.old_id = b.rev_text_id
```

Ми реалізували цю практику, використовуючи локальну інсталяцію програмного забезпечення Apache WebServer версії 2.2.25. Фактично, програма вже використовує практику «Переведення програми в режим сну» під час очікування HTTP-запиту. Ми змінили вихідний код WebServer, видаливши кожен виклик функції sleep () у тілі процедури обробки запиту. Ця модифікована версія відображає досліджуваний об'єкт без застосування цієї практики. Для бенчмаркінгу ми використовували утиліту ab (Apache Benchmark), що входить до пакета Web - Server, з такими параметрами:

ab -kc 50 -t 300 -n 5000000 http://localho.st/

Ця конфігурація видає до 5000000 запитів, максимум 50 одночасних запитів та обмеження часу 5 хвилин (300 секунд). Це дозволяє нам контролювати тривалість експерименту та отримувати статистику продуктивності: на відміну від попередньої практики, де кількість запитів була фіксованою, а час виконання - ні, у цьому випадку кількість запитів змінюється (веб-сервер не в змозі обробити 5 мільйонів запитів за 5 хвилин), але час виконання фіксований.

Для кожного сценарію (до, після) ми виконали 10 різних виконання. Під час кожного виконання ми збирали дані про використання ресурсів через CSV-вивід інструменту dstat , дані про використання енергії – через інструмент ESRV (на рівні ресурсів) та лічильник WattsUp PRO (на рівні системи), а також показники виконання програмного забезпечення – через API IEC. Ми ретельно перевіряли

часові позначки між нашими різними журналами, щоб забезпечити синхронізацію. Як показано на рисунку 2.1, усі журнали збиралися на окремому комп'ютері монітора , щоб мінімізувати накладні витрати на вимірювання на тестовому комп'ютері. З цієї причини лічильники енергії були електрично підключені до тестового комп'ютера, але канали даних лічильників (тобто USB-кабелі DAQ та WattsUp PRO) були підключені до комп'ютера монітора, де збиралися зразки даних. Що стосується вимірювань програмного забезпечення та даних про використання ресурсів, то виклики інструменту dstat та API IEC виконувалися на тестовому комп'ютері, але вихідні журнали CSV записувалися віддалено у спільну папку NFS, розташовану на комп'ютері монітора [19].

Аналіз даних було виконано за допомогою програмного забезпечення R для статистичних обчислень. Ми застосували такі методи аналізу даних (більшість з них описані в [36], посилання наведено в інших випадках):

- Описова статистика (наприклад, середнє значення, медіана)
- Тест Шапіро-Вілка на нормальність [37]
- Кореляційний аналіз з використанням рангового коефіцієнта Спірмена
- Знаково-ранговий тест Вілкоксона для оцінки впливу практики

Обчислення розміру ефекту

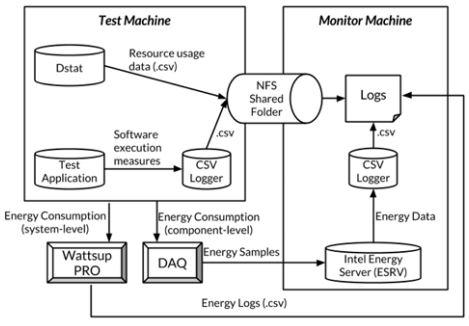

Рисунок 2.1 - Проведення експерименту

Ми обрали рівень значущості a = 0,05 для всіх наших тестів, тобто ми приймаємо 5% ймовірність помилки I типу (відхилення нульової гіпотези, коли

вона насправді істинна) [20]. Під час оцінки кореляцій, через велику кількість виконаних нами порівнянь (21 порівняння для аналізу на системному рівні, 168 для аналізу на ресурсному рівні), ми застосували поправку Бонферроні до нашого рівня значущості. Отже, на системному рівні ми маємо αc = 0,05/21 « 0,002 тоді як на рівні ресурсів ми маємо αc = 0,05/168 « 0,0003. Усі необроблені дані, звіти та R-скрипти, що відтворюють звіти, є загальнодоступними онлайн.

2.3 Паралельне та розподілене програмування в контексті енергоефективності

Паралельне програмування – це процес розбиття великої проблеми на менші для їх одночасного вирішення. У цьому розділі ми обговорюємо роботи, які емпірично оцінюють програми та алгоритми, що використовують паралельні обчислення. Підсумовано вплив пов'язаних стратегій управління потоками та застосованих коригувань на споживання енергії та продуктивність виконання для різних типів програм. Поля з від'ємними значеннями вказують на збільшення споживання енергії або часу виконання для відповідного джерела.

Проводячи емпіричне дослідження, визначили конфігурації та параметри, які можуть зменшити споживання енергії паралельними програмами [21]. Автори порівняли дев'ять існуючих стратегій управління енергоспоживанням, використовуючи стандартизовану системну архітектуру, операційну систему (ОС), інструмент вимірювання та п'ять пакетів бенчмарків. Стратегії (наприклад, налаштування частоти процесора, розгін, паралелізм та оптимізація компілятора) були запущені на 220 експериментальних конфігураціях та протестовані на 41 програмі, загалом понад 200 000 виконань. Отримані результати підкреслюють важливість ефективного паралелізації вихідного коду шляхом використання відповідної кількості потоків.

Щоб оцінити енергоефективність різних конструкцій керування потоками, провели емпіричне дослідження, порівнюючи три конструкції (тобто явне створення потоків, пул потоків фіксованого розміру та крадіжку роботи) на дев'яти

бенчмарках, коригуючи кількість потоків, рівень гранулярності завдання, розмір даних та характер доступу до даних. Автори зазначили, що споживання енергії конструкціями різниться в різних ситуаціях. Наприклад, явне створення потоків демонструє найнижче споживання енергії, коли йдеться про програми, пов'язані з вводом/виводом [22]. Для високопаралелізованих бенчмарків крадіжка роботи перевершує явне створення потоків та пул потоків фіксованого розміру, будучи на 30% енергоефективнішою; однак для більш серіалізованих бенчмарків крадіжка роботи є низькопродуктивною. Також автори помітили, що споживання енергії зростає зі збільшенням кількості потоків. Це відбувається доти, доки кількість потоків не досягне кількості ядер ПРОЦЕСОРА . Після цього автори виявили зниження споживання енергії для більшості протестованих програм.

Як видно, виграші та втрати щодо споживання енергії та продуктивності виконання можливі шляхом вибору відповідної кількості потоків. Наприклад, як Пінто та ін., так і Камбадур і Кім налаштували кількість потоків для оцінки своїх застосувань. Використовуючи різні конструкції керування потоками та різну кількість потоків, Пінто та ін. змінили споживання енергії з -50% до 30%, тоді як Камбадур і Кім зменшили його в середньому на 55%. Основна відмінність між двома роботами, яка може пояснити розбіжності в результатах, полягає в тому, що Пінто та ін. використовували програми Java, тоді як Камбадур і Кім використовували Java, а також програми на C з прапорцями оптимізації виконання компілятора.

Як показано вище, проведення експериментів з різною кількістю потоків або конструкцій керування потоками може допомогти практикам визначити конфігурації, які найімовірніше зменшать споживання енергії та час виконання їхніх програм [23]. Однак, взаємодія з користувачами є обов'язковою для визначення найкращих конфігурацій та параметрів шляхом експериментів, що робить процес вибору трудомістким та громіздким.

Щоб скористатися перевагами паралельної обробки, розробили алгоритми для мінімізації споживання енергії шляхом ефективного управління робочим навантаженням потоків. За словами Цая та ін., більшість методів динамічного

масштабування напруги та частоти (DVFS) – механізму, який налаштовує напругу ПРОЦЕСОРА для регулювання його частоти на основі поточного робочого навантаження – не призначені для роботи на багатопотокових процесорах; тому вони не можуть заощадити значно більше енергії [24]. З цією метою автори запропонували перетасовку потоків, спосіб поєднання таких методів, як міграція потоків та DVFS, для зменшення споживання енергії програмою без шкоди для її продуктивності під час виконання. Основна ідея перетасовки потоків полягає у визначенні потоків з однаковим ступенем критичності (повільне виконання потоків) за допомогою алгоритму прогнозування та відображенні їх під одним ядром за допомогою міграції потоків. Після цього алгоритм динамічно застосовує напругу для масштабування частоти ядер, що містять некритичні потоки (швидке виконання потоків).

Так само представили стратегію «ГЕРМЕС» ДЛЯ КРАДІЖКИ РОБОТИ, ЯКА ВИКОРИСТОВУЄ ПІДХІД «злодій-жертва» . Автори називають злодієм потік, який завершує свої завдання та краде роботу в інших потоків, так званих жертв. HERMES складається з двох основних алгоритмів: чутливого до робочого шляху та чутливого до робочого навантаження. Алгоритм , чутливий до робочого шляху, визначає темп потоку (швидкість виконання) на основі взаємозв'язку " злодій-жертва" ; це означає, що коли злодій краде у жертви, його темп встановлюється нижче (оскільки він завжди краде незначні завдання) і підвищується, коли жертва вичерпує роботу. Алгоритм , чутливий до робочого навантаження, - це механізм налаштування темпу виконання працівників на основі робочого потоку, який, за необхідності, коригує частоту ядра за допомогою DVFS для зменшення споживання енергії. Наприклад, якщо черга працівника порожня, він намагається вкрасти роботу в інших працівників; після цього він коригує частоту свого ядра відповідно до його робочого навантаження.

У вищезгаданих роботах, як перетасовка потоків, так і HERMES використовують методи DVFS з міграцією робочого навантаження для досягнення економії енергії [25]. Перетасовка потоків упаковує всі потоки з подібним рівнем критичності під певними ядрами, а потім відповідно налаштовує тактову частоту

ядер, тоді як HERMES змінює тактову частоту кожного ядра на основі поточного робочого навантаження потоків. Зокрема, підхід HERMES до крадіжки роботи призвів до незначної економії енергії, а саме: техніка перетасовки потоків призвела до економії енергії в середньому на 56%, як показано в таблиці 3. На завершення, HERMES забезпечив економію енергії з незначним зниженням продуктивності під час виконання (тобто 3-4%), тоді як перетасовка потоків досягла переконливої економії енергії без будь-якого шкоди для продуктивності під час виконання (див. таблицю 2.2).

Таблиця 2.2

Приблизний вплив обчислювальних потужностей на енергоефективність

Назва інструмента	Орієнтовний фокус оптимізації	Вплив (у %)	Втрата точності
		Енергія	Продуктивність
GREEN	Обчислення та завершення циклів	14 (сер.)	21 (сер.)
—	Мемоїзація функцій	74 (сер.)	79 (сер.)
Parrot	Оптимізація ділянок імперативного коду	66 (сер.)	56 (сер.)
Назва інструмента	Орієнтовний фокус оптимізації	Вплив (у %)	Втрата точності
		Енергія	Продуктивність
GREEN	Обчислення та завершення циклів	14 (сер.)	21 (сер.)
—	Мемоїзація функцій	74 (сер.)	79 (сер.)
Parrot	Оптимізація ділянок імперативного коду	66 (сер.)	56 (сер.)
Chisel	Операції обчислювального ядра	9–20	Chisel
EnerJ	Обчислення, зберігання даних та алгоритми	10–50	EnerJ
Axilog	Частини вихідного коду	54 (сер.)	Axilog
—	Вибрана група завдань	—	—
DCO SCORPIO	Вибрана група завдань	56 (сер.)	DCO SCORPIO
Chisel	Операції обчислювального ядра	9–20	Chisel

Наближені обчислення – це підхід, що дозволяє пожертвувати точністю обчислень (коли програма це допускає) для підвищення продуктивності виконання або економії енергії. Наприклад, такі методи, як перфорація циклів , дозволяють користувачам керувати компромісами між продуктивністю виконання та точністю на основі бажаної якості виводу. У цьому розділі ми обговорюємо дослідження,

розділені на фреймворки програмування, мемоізацію, розширення на основі анотацій та розширення на основі директив. У таблиці 4 ми
підсумовуємо відповідні роботи.

Для своїх досліджень, запропонували енергоефективні програмні фреймворки для досягнення економії енергії за допомогою наближених обчислень. Зокрема, Місайлович та ін. запропонували Chisel, фреймворк оптимізації, який діє автоматизовано, вибираючи наближені операції ядра, що призводять до оптимізації енергії, надійності та точності. Бек і Чілімбі запропонували GREEN, фреймворк, спрямований на оптимізацію дорогих циклів і функцій, враховуючи визначені користувачем вимоги до якості обслуговування (QoS). GREEN досягає економії енергії за рахунок наближених обчислень для функцій і раннього завершення циклу. Крім того, він враховує компроміси між QoS та споживанням енергії, застосовуючи методи наближеного програмування лише тоді, коли вимоги QoS виконані. Спільним елементом як GREEN, так і Chisel є порівняння точних і наближених випадків для розрахунку надійності результатів. Однак, що відрізняє GREEN від Chisel , так це періодична вибірка QoS під час виконання для коригування методів наближення та моделі QoS для відповідності цільовим вимогам. На відміну від ЕКОЛОГІЧНО ЧИСТОГО ВАРІАНТУ, Chisel використовує приблизну пам'ять своєї робочої платформи для збільшення економії енергії, жертвуючи частиною своєї якості продукції (див. таблицю 2.2).

Запропоновано відповідно EnerJ, перетворення Parrot та Axilog ; усі вони є розширеннями, які досягають економії енергії шляхом виконання приблизно анотованих частин вихідного коду. Зокрема, EnerJ - це розширення на основі Java, що має функціональність ручного анотування для визначення приблизного або точного вибору даних для програми. Оголошуючи змінні та об'єкти як приблизні, EnerJ відображає їх у приблизні дані пам'яті . і генерує низьковитратний енергоємний код, використовуючи наближені операції та алгоритми [26]. Перетворення Parrot - це модель нейронної мережі, яка ідентифікує імперативні області коду та передає їх нейронному процесору, а не центральному процесору, для підвищення енергоспоживання та продуктивності виконання. Axilog - це

розширення Verilog, яке створює короткі та високорівневі анотації для повного контролю та управління наближеним обладнанням. На відміну від ручного підходу до анотування, як це роблять EnerJ та Parrot , Axilog використовує алгоритм аналізу інтерфейсу релаксації для автоматизації процесів апроксимації на основі вибору розробника. Усі три системи анотацій пропонують механізм безпеки для ізоляції наближених від точних частин коду, тим самим гарантуючи основну функціональність програми.

На відміну від GREEN, Chisel та Axilog, в EnerJ та Parrot розробники є власниками застосованих методів апроксимації, вибираючи, які частини коду мають бути виконані з точністю. Таким чином, EnerJ та Parrot можуть допомогти практикам краще зрозуміти компроміси між енергетичною апроксимацією та їхнім вибором проектних рішень.

Інший підхід до застосування методів апроксимації до обчислювальних завдань полягає в залежності від їхнього рівня значущості (рівня важливості або критичності) [27]. У цьому напрямку запропонували підходи на основі директив шляхом розширення OpenMP, які використовують методи апроксимації для зменшення споживання енергії додатками. У роботі запропонували модель програмування, спрямовану на досягнення найвищого рівня точності для додатка відповідно до визначеного користувачем енергетичного бюджету. Тому автори представили систему виконання, яка відповідає за вибір відповідних конфігурацій (тобто кількості ядер, тактової частоти та коефіцієнта точності) для певного розміру вхідних даних. Відповідні конфігурації виводяться моделлю, яка навчена ідентифікувати конфігурації, що забезпечують найвищу можливу точність виводу для заданого енергетичного бюджету [28]. Інший підхід - DCO/SCORPIO, фреймворк, який підтримує автоматизований аналіз для визначення рівня значущості коду. DCO/SCORPIO досягає цього, використовуючи інтервальну арифметику та алгоритмічне диференціювання для кількісної оцінки значущості певних обчислень для певного вхідного значення. Цей вихідний результат, у свою чергу, використовується моделлю, подібною до OpenMP, для класифікації обчислень у

групах завдань відповідно до їхнього рівня значущості. Таким чином, він надає приблизні методи на основі рівня значущості кожної групи.

DCO/SCORPION та робота відрізняються тим, що перший зменшує споживання енергії застосунком, тоді як другий досягає найвищої можливої якості виводу в межах певного порогу споживання енергії. Крім того, DCO/SCORPIO звільняє руки користувача, вибираючи важливість обчислювальних завдань, вимагають участі програміста для введення директив для критично важливих частин коду, які не допускають неточності.

Мемоізація. Інший підхід до економії енергії за допомогою наближених обчислень полягає у використанні мемоізації для зберігання результатів викликів ресурсомістких функцій. Розробили модель продуктивності для вибору та мемоізації обчислювально ресурсоємних функцій з фінансових додатків та бенчмарку JavaGrande. Порівняно з вищезазначеними підходами, мемоізація, здається, має найвищу економію енергії та продуктивність виконання (див. таблицю 2.2). Однак автори не застосовували свій метод до реальних додатків.

Аналіз вихідного коду – це процес тестування, який зосереджений на виявленні дефектів та вразливостей у комп'ютерній програмі перед її розгортанням. У цьому розділі ми обговорюємо роботи з динамічного аналізу вихідного коду, спрямовані на виявлення помилок та гарячих точок, пов'язаних з енергоспоживанням, шляхом тестування комп'ютерної програми в режимі реального часу. У контексті аналізу вихідного коду ми не знайшли доступних інструментів для статичного аналізу коду, які б надавали правила для аналізу вихідного коду перед виконанням [29]. У таблиці наведено роботи з аналізу вихідного коду та надано інформацію про цільову платформу, кореляцію вимірювань енергії на різних рівнях деталізації програмного забезпечення та коефіцієнт допустимої помилки.

GreenAdvisor - це системний профайлер, який прогнозує поведінку, продуктивність виконання та зміни, пов’язані з енергоспоживанням, у програмі. Щоб передбачити зміни, пов'язані з енергоспоживанням, GreenAdvisor порівнює кількість системних викликів у поточній версії програмного забезпечення та

попередній. Якщо споживання енергії збільшується, то він точно визначає енергетичні гарячі точки, які спричинили зміни, допомагаючи розробникам аналізувати та розуміти наслідки своїх рішень.

Таблиця 2.3

Інформація про аналогічні роботи з аналізу вихідного коду

Назва інструмента	Цільова платформа	Кореляція вимірювання енергії	Похибка (у %)
Eprof	Android та Windows OS	Процеси, потоки, системні виклики, підпрограми	<6
eLens	Android	Повна деталізація вихідного коду	10
GreenAdvisor	Android	Підпрограми, системні виклики	—
PEEK	Вбудовані системи (Embedded)	Системи на рівні функцій	—
SEEDS	Будь-яка платформа на базі Java	Користувач може сам вибрати частину для аналізу	—
—	Смартфони	Рівень функцій	—

Для діагностики вузьких місць в енергоспоживанні на рівні вихідного коду, представили Eprof, інструмент моделювання енергоспоживання на основі системних викликів для додатків для смартфонів. Для досягнення високої точності вимірювань споживання енергії Eprof включає два наступні компоненти. По-перше, він використовує кінцеві автомати для моделювання різних станів та переходів енергоспоживання для окремих апаратних компонентів та смартфона в цілому. Потім, для кожного апаратного компонента, він запускає набір бенчмарків, що складається з додатків для збору різних системних викликів та їх переходів енергоспоживання. Після цього він генерує правила для кінцевих автоматів, інтегруючи зібрані знання з виконуваних додатків. Для оцінки споживання енергії на рівні вихідного коду, Eprof порівнює підпрограми (блоки коду) зі слідами системних викликів. В результаті автори виявили, що додатки, що використовують сторонні процеси, як правило, мають збільшення споживання енергії на 65-75%.

Загалом, вищезгадані інструменти використовують різні моделі оцінки енергії для представлення вимірювань енергії. Наприклад, Eprof використовує модель, яка розраховує споживання енергії програмою на звичайному рівні та різними апаратними компонентами, тоді як GreenAdvisor порівнює споживання

енергії різними версіями продукту та вказує на системні виклики, які спричинили зміну.

Рекомендували інструменти для динамічного аналізу вихідного коду. Як запропонували підходи до програмування з урахуванням енергоспоживання, щоб допомогти розробникам на етапі впровадження, надаючи підказки, пов'язані з енергоспоживанням. Honig та ін. представили Proactive Energy-aware Development Kit (PEEK), тоді як Manotas та ін. просували фреймворк Software Engineer's Energy-optimization Decision Support (SEEDS) [30]. Періодично обидва підходи аналізують вихідний код, що розробляється, та безперешкодно створюють багато різних екземплярів з поточного вихідного коду для визначення оптимальних модифікацій коду. Однак суттєва різниця між цими двома підходами полягає в тому, що підказки PEEK, пов'язані з енергоспоживанням, пов'язані з механізмами управління живленням (наприклад, стан сну, DVFS, стан очікування, логіка програмного коду, бібліотеки та прапорці оптимізації виконання компілятора), тоді як пропозиції SEEDS обмежуються бібліотеками колекцій Java (JCL), алгоритмами або рефакторингом частин вихідного коду. Крім того, SEEDS пропонує розробникам можливість встановити діапазон блоків коду для аналізу, тоді як PEEK аналізує вихідний код на рівні деталізації функцій.

Іншим способом виявлення помилок, пов'язаних з енергоспоживанням, є використання автоматизованої структури генерації тестів. Вищезгадане дослідження вказує на пов'язані з енергоспоживанням гарячі точки у чотирьох категоріях програм для смартфонів, а саме: (1) апаратні ресурси, (2) переходи з режиму сну в стан сну, (3) фонові служби та (4) дефектна функціональність. Спочатку запускається процес виявлення для пошуку можливих взаємодій користувача за допомогою графіків потоку подій. Потім запропонована структура генерує тестові випадки для фіксації сценаріїв взаємодії та, згодом, для виявлення гарячих точок енергоспоживання. Поряд з ідентифікацією гарячих точок енергоспоживання, інструмент видає звіти про тестування для розробників [31]. Однак, порівняно, робота не надає підказок щодо оптимізації енергоспоживання, а лише знаходить частини програми, що витрачають енергію.

Рядок за рядком. Для вимірювання споживання енергії програмами на різних рівнях деталізації програмного забезпечення та підвищення обізнаності про енергоспоживання на етапі розробки, Хао та ін. запропонували eLens. eLens оцінює споживання енергії за допомогою аналізу програми та моделювання енергоспоживання для кожної інструкції. Використовують аналіз програми для отримання інформації, пов'язаної з виконанням, такої як байт-код або виклики API з різних компонентів смартфона (наприклад, ПРОЦЕСОР, ОПЕРАТИВНА ПАМ'ЯТЬ, GPS, 3G). Потім зібрана інформація передається до компонента моделювання енергоспоживання для кожної інструкції для оцінки споживання енергії програмою. Таким чином, eLens забезпечує вимірювання енергії на різних рівнях деталізації: програма, метод, клас, шлях та рядки вихідного коду. Порівняно з усіма представленими інструментами, eLens є єдиним, який забезпечує вимірювання енергії на всіх рівнях деталізації вихідного коду. Таким чином, підвищується обізнаність про енергоспоживання, надаючи детальну інформацію про споживання програмами.

Мови програмування пропонують набір інструкцій, які дозволяють користувачам використовувати системні ресурси для вирішення проблеми. Мови відрізняються за функціями та способом розподілу обчислювальних ресурсів [32]. У цьому розділі ми розглядаємо емпіричні дослідження, що досліджують вплив мов програмування на споживання енергії та продуктивність під час виконання. У таблиці 6 перераховано пов'язані роботи з точки зору споживання енергії, продуктивності під час виконання, використаного прапора оптимізації та тестових випадків. Ми підсумовуємо пов'язані результати у зведеному списку, що знаходиться в таблиці 2.4 , середніх значень, розрахованих нами .

Дослідження, представлені в цьому розділі, базуються на різних експериментальних платформах. Зокрема проводили свої тести на робочій станції, використовували вбудовану систему, проводили свої експерименти на смартфоні. Однак використовували подібні параметри тестування. Крім того , Абдулсалам та ін. також порівняли енергетичні наслідки чотирьох методів розподілу пам'яті

(тобто malloc, new, array та vector), де вони представили malloc як найбільш енергоефективний та ефективний під час виконання.

Таблиця 2.4

Вплив мов програмування на енергоефективність

Порівняння (X до Y)	Вплив (сер. у %)	Прапор оптим.	Середовище виконання	Вибраний додаток
	Енергія	Продуктивність		
C++ до C	8.4	8.67	-O3	—
C++ до Java	47.4	38.12	-O3	—
C++ до Python	166.28	195.17	-O3	—
C до Java	38.39	29.08	-O3	—
C до Python	106.4	195.5	-O3	—
Java до Python	195.87	194.77	-O3	—
C++ до Java	166.14	159.13	-O3	Dalvik
C++ до Java	Ідентично	Ідентично	-O3	ART
C до C++	Ідентично	3.38	—	—
ARM-assembly до C	15.38	10.52	—	—
ARM-assembly до Java	58.84	90.90	—	—

У своєму експерименті використали Native Development Kit . набір інструментів для виконання нативного коду, такого як C та C++, у застосунках Android. Отриманий результат, полягає в тому, що C та C++ досягли значної економії енергії, а також скоротили час виконання порівняно з іншими мовами програмування. Також обидві роботи показали, що прапор оптимізації компілятора середовища виконання -03 мав найбільшу економію енергії та підвищив продуктивність виконання для застосунків для робочих станцій та смартфонів. Більше того, для застосунків Java показали, що використання середовища виконання Android замість середовища виконання Dalvik сприяло результатам енергоспоживання та продуктивності виконання, подібним до реалізацій на C та C++.

У контексті вбудованих систем провели експеримент для порівняння впливу на енергоспоживання та продуктивність чотирьох алгоритмів сортування, написаних на ARM-асемблерних мовах, C та Java. Виконуючи свої експерименти на Raspberry Pi , Автори показали, що реалізації ARM-складання досягли найбільш

енергоефективних результатів, а саме реалізації на C та Java. Аналогічно, представили Java як найбільш енергоємну серед вибраних мов програмування.

Обмеженням, яке ми спостерігали для всіх обговорюваних робіт, є відсутність інформації щодо версій компіляторів, інтерпретаторів, модулів та бібліотек, що використовувалися в експериментах [33]. Крім того, ми не знайшли жодного дослідження, яке порівнює споживання енергії та вплив продуктивності виконання однієї й тієї ж програми на різні версії компілятора, механізму виконання або інтерпретатора.

Таблиця 2.5

Конфігурації мов програмування

Мови програмування	Прапори оптимізації	Цільові платформи	Тестові випадки (Test Cases)
C, C++, Java та Python	-O{1,2,3}	Серверна система	Fast Fourier (Швидке перетворення Фур'є), Linked List, Quick Sort
ARM assembly, C та Java	—	Вбудована система	Bubble (Бульбашка), Merge Quick (Злиття), Counting (Підрахунок)
C, C++ та Java	-O{1,2,3}	Android-пристрої	Fibonacci, Tower of Hanoi (Ханойська вежа), Pi calculation (Обчислення числа Пі)

Структура даних – це спосіб організації, керування та зберігання даних для подальшої обробки або аналізу. Цей розділ складається з емпіричних досліджень та інструментальної підтримки структур даних. У розділі «Емпіричні дослідження » ми обговорюємо роботи, спрямовані на визначення того, які структури даних є найбільш енергоефективними для конкретних випадків. Щодо інструментальної підтримки , ми показуємо інструменти, які інформують фахівця, яку структуру даних обрати для зменшення споживання енергії [34]. У таблиці 2.5 підсумовано роботи, присвячені інтерфейсу збору структур даних та бібліотеці, споживанню енергії та протестованим програмам для відповідного джерела.

Експериментальні дослідження. Проводячи експериментальні дослідження, виявили деякі позитивні та негативні випадки щодо споживання енергії різними

структурами даних [35]. Автори оцінили споживання енергії структурами даних з різних інтерфейсів, але здебільшого з Java Collection Framework (JCF).

Проведено експериментальне дослідження для оцінки енергоефективності різних методів інтерфейсів JCF , таких як пошук, ітерація, видалення та вставка. Завдяки ручній заміні структур даних у додатках автори отримали значну економію енергії. Оскільки автори вищезгаданої роботи повідомляють про великий список результатів, ми порівняли споживання енергії найбільш та найменш енергоефективним інтерфейсом і представляємо структуру даних з найбільш енергоефективними методами для кожного інтерфейсу.

Досліджуючи використання пам'яті та аналізуючи трасування байт-коду додатків Android, оцінили споживання енергії різними типами колекцій. Окрім JCF, автори використовували структури даних з колекцій Apache Commons (ACC) та Trove [36]. В результаті, автори показали, що споживання енергії починає розходитися між структурами даних лише тоді, коли кількість елементів, які вони містять, перевищує 500. Крім того, автори зазначили для структур даних з примітивними типами даних (знайдених у колекції Trove), що, хоча вони споживають менше пам'яті, ніж об'єкти, у більшості випадків вони є менш енергоефективними.

Використовували 13 потокобезпечних та три непотокобезпечні реалізації JCF. Автори протестували вищезазначені структури даних, використовуючи різні конфігурації, такі як кількість потоків, початкова ємність та коефіцієнт навантаження. В результаті автори помітили, що правильний вибір структури даних та кількості потоків (для більшості потокобезпечних та непотокобезпечних реалізацій) може зменшити споживання енергії додатками. Наприклад, коли автори замінили екземпляри HashTable на ConcurrentHashTableV8 , у реальних бенчмарках вони досягли значної економії енергії.

Загалом показали, що реалізації однакових методів (наприклад, додавання, видалення, пошук) по-різному впливають на споживання енергії структурами даних. У таблиці ми бачимо, що досягли найбільшої економії енергії у своїх експериментах [37]. Однак, порівняно використовували мікро-бенчмарки, а не

реальні програми. Пінто та ін. досліджували вплив реальних програм на енергію шляхом (i) заміни не-потокобезпечних структур даних на потокобезпечні та (ii) зміни кількості потоків. Це допомогло їм досягти суттєвої економії енергії. Крім того, результати Хасана та ін. показують негативний вплив нерозумного вибору структури даних, який може суттєво вплинути на споживання енергії.

Прогнозування найбільш енергоефективних структур даних для заданої проблеми може зменшити споживання енергії. У цьому представили GreenC5, інструмент, який може передбачати, які структури даних серед колекції Copenhagen Comprehensive Collection Classes for C# (C5) можуть зменшити споживання енергії застосунками на основі робочого навантаження системи. GreenC5 складається з прогнозного алгоритму, заснованого на моделях машинного навчання та нейронних мереж. Як показано в таблиці 8 , досягли економії енергії для реальних застосунків.

Щодо підтримки інструментів, ми спостерігаємо кілька недоліків, таких як обмежена кількість типів колекцій (НАЧАЛЬНИЙ ЕЛЕМЕНТ доступний лише для черги) або зосередженість на певних колекціях (GreenC5 використовує лише колекцію C5). Однак ці інструменти можуть допомогти розробнику вручну вибирати та експериментувати з різними структурами даних для досягнення достатньої економії енергії. Більше того, подальша підтримка різних типів структур даних та різних колекцій може зробити такі інструменти більш корисними та привабливими для розробників.

Найкращі практики кодування – це набори правил, формально чи неформально, встановлених різними спільнотами програмістів, які допомагають фахівцям-програмістам покращувати якість програмного забезпечення. У цьому розділі ми обговорюємо роботи з емпіричної оцінки, які досліджують енергоспоживання практик кодування [38]. У таблиці підсумовано результати, визначені з пов'язаних робіт як «хороші» та «погані» практики кодування. Термінами «хороші» та «погані» ми називаємо практики кодування, які можуть позитивно чи негативно впливати на читабельність, зручність обслуговування, ефективність та зручність використання програми відповідно.

Щодо вбудованих систем, оцінили продуктивність виконання, споживання енергії, використання пам'яті та розмір коду п'яти блокових шифрів (тобто RC6, Rijndael, Serpent, Twofish та XTEA) на процесорі StrongARM SA-1100. Автори змінили існуючий вихідний код алгоритмів шифрування (щоб зменшити кількість рядків коду) шляхом:

- Заміна макросів викликами функцій .
- Використання розгортання циклу у функціях шифрування та дешифрування.
- Заміна T-пошуку прямими та зворотними S-блоковими таблицями та зменшення їх розмірів.

Їхні результати показують, що блокові шифри XTEA та RC6 (які мали найменший розмір коду) були найбільш енергоефективними, пропонували найкращу продуктивність під час виконання та використовували менше основної пам'яті для завдань шифрування та дешифрування.

У своєму експерименті Тоніні та ін. досліджували енергоефективність найкращих практик розробки Android. Їхні результати показують, що правильне використання циклу for та методів отримання/установки може покращити споживання енергії. Спочатку автори провели експерименти, використовуючи різні варіації циклу for, тобто for-each , коли умова завершення циклу і) обчислюється на кожній ітерації, та іі) коли вона передається як змінна. Крім того, автор оцінив сценарії з методами отримання/установки та без них для доступу до полів класу [39]. Їхні результати показують значну економію енергії завдяки використанню змінної як умови завершення циклу та доступу до змінних класу без використання функцій отримання/установки.

Так само перевірили такі практики, як групування HTTP- запитів із певним розміром та використанням пам'яті, а також поради щодо продуктивності. Зокрема, для порад щодо продуктивності автори перевірили методи кодування, отримані з форуму розробників Android . Застосування практик кодування допомогло досягти значної економії енергії, як показано в таблиці 2.6. Практика уникнення обчислення довжини структури даних у циклі виявилася корисною, оскільки наявність умови

завершення циклу в змінній економить енергію, обходячи обчислення довжини на кожній ітерації. Практика прямого доступу до поля також виявилася корисною, оскільки не потрібен додатковий виклик функції від системи, на відміну від випадку, коли значення поля отримується через виклик методу. Нарешті, практика статичного виклику виявилася енергоефективною, оскільки виклик методу статично економить енергію, оскільки уникає накладних витрат на пошук для виклику методів через існуючий об'єкт.

Android пропонує різноманітні виклики інтерфейсу прикладного програмування (API) , і якщо їх неефективно використовувати, вони можуть сприяти збільшенню споживання енергії. У своєму дослідженні провели аналіз 55 Android-додатків з різних доменів і перерахували найбільш енергоефективні методи API . Крім того, автори запропонували список практик, які можуть забезпечити економію енергії завдяки ефективному використанню викликів API [40]. Щоб отримати свої результати, автори співвіднесли часові позначки трасування виконання методів з вимірюваннями споживання енергії. Після аналізу своїх результатів вони визначили 133 енергоємні API з загальної кількості 807. З енергоємних API 61% пов'язані з графічним інтерфейсом користувача та маніпулюванням зображеннями, тоді як решта 39% належать до категорії баз даних. На завершення автори наголосили, що непотрібне оновлення подань (наприклад, перемальовування подання після отримання нових даних) та віджетів може споживати значну кількість енергії. Крім того, вони наполегливо рекомендують користувачам уникати реляційних баз даних, таких як SQLite , коли це не має першочергового значення рекомендували та оцінювали дві найкращі практики. Обрані практики полягали в тому, щоб перевести програму в режим сну, тобто перевести процеси або потоки в стан сну, якщо вони очікують на операції вводу/виводу або більше не активні, та використовувати ефективні запити, тобто уникнути використання дорогих енергоємних операцій, таких як упорядкування або індексування, коли вони не потрібні. В результаті автори досягли енергоефективності, використовуючи обидві практики, а також збільшили продуктивність виконання.

Результати показують, що використання для заданого розміру довжини сприяє економії енергії від 36% до 52% згідно з Тоніні та ін. та Лі та Халфондом відповідно. Більше того, в обох роботах автори уникали використання геттерів/сеттерів та заощаджували енергію на 24-27% та 30-35%. Хоча в обох дослідженнях використовувалися мікробенчмарки , їхні результати пропонують різний масштаб економії енергії.

Таблиця 2.6

Інформація про аналогічні роботи

Набір бенчмарків	**Цільова платформа**	**Кореляція енергії з...**	**Рівень оптимізації**
GBench	Linux	Апаратними компонентами	Переміщення даних у пам'яті, розмір блоку, кількість ядер
ALEA	Linux	Базовими блоками коду	Обмеження потужності (Power capping), DVFS, оптимізація компілятора, дроселювання потоків (thread throttling)
AxBench	Linux	Апаратними компонентами	Вихідний код через наближення (approximation)
PowerBench	TinyOS	Кожним вузлом тестового стенда	—

Вони використовували різні апаратні пристрої, різні версії Android та різні інструменти для отримання своїх вимірювань енергії. З таблиці 2.6 також можна побачити, що Гросшадль та ін. досягли значної економії енергії (тобто 179%), використовуючи відповідні методи кодування. Однак слід зазначити, що автори порівнювали блокові шифри один з одним, а не оригінальну та оптимізовану версії.

Підсумовуючи, існують можливості для покращення енергоспоживання та продуктивності виконання навіть шляхом застосування незначних змін у коді, таких як передача змінної в умові завершення циклу або статичний виклик методу. Більше того, правильний вибір A , який вимагає менше системних ресурсів, також може зменшити споживання енергії. Крім того, покращили енергоспоживання та продуктивність виконання, зменшивши використання дорогих операцій з базою даних (коли вони не були обов'язковими) та переводячи програми в режим сну (коли вони не виконували жодних дій).

2.4 Висновки по розділу

У другому розділі було досліджено методи та інструменти розробки енергоефективних програмних систем. Розглянуто архітектурні та дизайнерські підходи, які спрямовані на зменшення енергоспоживання програмного забезпечення. Проаналізовано методи оптимізації виконання програм, що дозволяють підвищити ефективність використання обчислювальних ресурсів. Окрему увагу приділено паралельному та розподіленому програмуванню як засобам підвищення продуктивності з одночасним зниженням енергетичних витрат. У результаті визначено ефективні підходи до створення енергоефективних програмних систем.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕНЕРГОЕФЕКТИВНОСТІ ПРОГРАМНИХ РІШЕНЬ

3.1 Інструменти вимірювання та тестування енергоспоживання програм

Тут ми обговорюємо низку інструментів для вимірювання та тестування енергії та споживання енергії програмним забезпеченням після розробки застосунку. Ми розділяємо збірку праць на розділи «Тестові показники» та «Інструменти моніторингу».

Бенчмарки – це інструменти, що складаються з двох основних компонентів: (i) інструменту профілювання, який відповідає за отримання інструкцій, що використовуються під час конкретного виконання, та (ii) бенчмарку продуктивності, який генерує робочі навантаження для системи. Вищезазначені компоненти разом виконують вимірювання споживання енергії. Підсумовано зібрану інформацію щодо цільової платформи, кореляції вимірювань енергії з апаратними або програмними компонентами, застосованих оптимізацій та пов'язаних ресурсів.

PowerBench - це масштабована тестова інфраструктура, яка паралельно проводить порівняльний аналіз та отримує дані про споживання енергії з різних бездротових сенсорних вузлів кластера. Для цього PowerBench використовує спеціальні апаратні та програмні компоненти, щоб забезпечити автономну обробку, аналіз, налагодження та візуалізацію отриманих вимірювань потужності. Особливості такого підходу можуть допомогти розробникам виявляти коливання та аномалії споживання енергії в кластерах та складних середовищах Інтернету речей.

Запропонували «Зелений бенчмарк» (GBench) – підхід, який використовує змінну навантаження LiNpAc K який створює різноманітні робочі навантаження для оцінки споживання енергії системою. Автори протестували GBench з різними конфігураціями, такими як розмір блоку, робочі навантаження, кількість ядер та

швидкість доступу до пам'яті. Як наслідок, вони виявили кореляцію між споживанням енергії та продуктивністю виконання для другого рівня (L2) промахів кешу на певних робочих навантаженнях.

Запропонували Абстрактний енергетичний облік (ALEA) – високоточний (середнє значення похибки 1,4%) портативний інструмент, який отримує вимірювання енергії з будь-якої мікропроцесорної архітектури. По суті, ALEA має інструмент детального профілювання енергії, який отримує вимірювання з базових блоків коду. Для оцінки автори використовували відомі пакети бенчмарків, такі як SPEC 2000 , СПЕЦИФІКАЦІЯ , ОМП , тощо, а також проаналізували зв'язок між споживанням енергії базовими блоками коду та обсягом доступу до кешу, щоб визначити точки підвищеного енергоспоживання. В результаті, використовуючи ALEA, автори досягли 37% економії енергії за деякими згаданими тестами завдяки різним енергоефективним стратегіям та коригуванням, наприклад, дроселювання паралельності, пакування потоків.20 Крім того, вони виявили сильну кореляцію між споживанням енергії та швидкістю доступу до кешу .

AxBench – це набір бенчмарків, який підтримує тести в різних галузях, таких як фінанси, обробка сигналів, обробка зображень, машинне навчання тощо, спрямований на оцінку часу роботи та енергоефективності систем шляхом використання методів апроксимації. Зокрема, методи апроксимації, що підтримуються AxBench. складаються з блоків перфорації циклу та нейронної обробки . AxBench отримує вимірювання енергії процесора, ГРАФІЧНОГО ПРОЦЕСОРА та обладнання Axilog. AxBench пропонує (1) функцію тестування різних рівнів обчислювального стеку (програмного та апаратного забезпечення), (2) різні тестові вхідні дані та (3) показники якості, специфічні для програми, тобто середню відносну похибку для числового виводу, коефіцієнт помилок для логічного результату та різницю в зображенні. Однак, щоб виконати бенчмаркінг, користувач повинен визначити та вручну позначити області коду, які можуть допускати неточність.

Основна відмінність між GBench a ALEA полягає в тому, що ALEA

співвідносить вимірювання енергії з базовими блоками коду, тоді як GBench відображає споживання енергії всієї програми на апаратні компоненти. AxBench оснащений бенчмарками для процесора та ГРАФІЧНОГО ПРОЦЕСОРА , що мають на меті забезпечити детальне розуміння їхнього впливу на енергоспоживання та продуктивність під час виконання. На відміну від вищезазначених бенчмарків, PowerBench є єдиним, хто оцінює споживання енергії готоподібною інфраструктурою та кластером.

Для визначення споживання енергії комп'ютерною системою наразі існують два підходи: (1) непрямі вимірювання енергії за допомогою моделей оцінки або лічильників продуктивності, або (2) прямі вимірювання за допомогою апаратних аналізаторів енергії та датчиків. У цьому розділі ми обговорюємо інструменти, які використовують непрямі та прямі підходи для виконання вимірювань, деякі з представлених нами інструментів аналізують запущені програми або системні виклики для оцінки споживання енергії. Однак, порівняно з інструментами аналізу вихідного коду інструменти моніторингу енергії повідомляють лише про споживання енергії застосунком, не вказуючи на гарячі точки енергії та не надаючи підказок щодо покращення виявлених недоліків. Таблицю 3.1 презентує обговорюване програмне забезпечення та інструменти моніторингу енергоспоживання обладнання.

У таблицях наведено різноманітну інформацію, таку як назви інструментів, цільова платформа, типи вимірювань, інтервали вибірки та середні показники помилок.

Ми використовуємо термін «програмні засоби моніторингу енергоспоживання» для програмних аналізаторів, які використовують лічильники продуктивності або моделі оцінки для вимірювання споживання енергії запущеними програмами. Ми аналізуємо засоби моніторингу щодо їхніх функцій, обмежень, архітектури (моделі оцінки енергоспоживання) та підтримуваних ОС.

Таблиця 3.1

Інформація про аналогічні роботи, пов’язані з програмними засобами моніторингу

Назва інструмента	Цільова платформа	Тип вимірювання	Частота дискретизації (мс)	Медіанна похибка (%)
Jalen	Linux	Енергія	500	—
PowerAPI	Linux	Енергія	500	0.5–3
jRAPL	Linux	Енергія	1	1.13
Jolinar	Linux	Енергія	500	3
RAPL	Linux	Енергія	1	3
SoCWatch	Windows, Linux, Android	Потужність	1–1000	—
PowerGadget	Windows & Linux	Потужність	1–1000	—
JouleMeter	VMS & Windows	Потужність	1000	5
VMeter	VMS	Потужність	—	6
BitWatts	VMS	Енергія	500	2
PowerBooter	Android	Потужність	—	0.8
GreenOracle	Android	Енергія	—	10
AEP	Android	Потужність	—	—
PETtA	Android	Енергія	1000	0.04

Крім того, ми класифікуємо програмні засоби моніторингу енергоспоживання відповідно до платформи, на яку вони орієнтовані, на три категорії: (1) робочі станції та сервери, (2) ВІРТУАЛЬНІ СИСТЕМИ КЕРУВАННЯ (VRS) та (3) смартфони. Робочі станції та сервери. Функція Running Average Power Limit (RAPL) контролює та контролює споживання енергії за допомогою лічильників продуктивності. Використовуючи псевдофайлову систему ядра Linux (sysfs), RAPL надає доступ до підсистем ядра, апаратних пристроїв та інформації про драйвери пристроїв з ядра простору користувача, що дозволяє оцінити споживання енергії програмним забезпеченням. Представили JRAPL , фреймворк, що поєднує RAPL та Java Native Interface виміряти споживання енергії ПРОЦЕСОРОМ, ОПЕРАТИВНОЮ ПАМ'ЯТТЮ та пакетом програмного забезпечення компоненти для програми Java. Окрім jRAPL представили інструменти PowerGadget та SoCWatch відповідно. Обидва інструменти використовують RAPL для отримання даних про споживання енергії з лічильників продуктивності ПРОЦЕСОРА. PowerGadget отримує показники живлення пакетів, що надаються процесором ТА графічним ПРОЦЕСОРОМ , і може

бути інтегрований у користувацьку програму через API C++. SoCWatch надає інформацію, пов'язану з живленням, для станів C та P ПРОЦЕСОРІВ та графічних процесорів. Усі обговорювані інструменти на основі RAPL мають високу частоту дискретизації та можуть отримувати значну кількість вибірок за мілісекунду. Однак інструменти, що включають RAPL , працюють за певних апаратних обмежень, таких як певна архітектура мікропроцесора. Наприклад, PowerGadget сумісний лише з ПРОЦЕСОРАМИ Intel другого покоління і ще не підтримується для таких архітектур, як Skylake, Broadwell та Haswell.

Для оцінки споживання енергії застосуванням запропонували низку інструментів. Запропонували Агент Java, підключений до програми, який збирає вимірювання енергії після ініціалізації JVM . Jalen вимірює вибрані методи та класи. Крім того, він надає вимірювання для конкретних апаратних компонентів (наприклад, ПРОЦЕСОРА та ЖОРСТКОГО ДИСКА) та оцінює споживання енергії програмним забезпеченням, аналізуючи виконувані інструкції Java. Jolina r - це інструмент на основі Java для моніторингу споживання енергії на рівні процесу. За словами його розробників, інструмент вимірює споживання енергії певними апаратними компонентами, такими як ПРОЦЕСОР, ОПЕРАТИВНА ПАМ'ЯТЬ та ЖОРСТКИЙ ДИСК. Однак, як Jalen, так і Jolinar використовують старий енергетичний модуль Intel, який не підтримується ядром Linux версій 3.10 і вище, якщо не вимкнено Intel p_state .PowerAPi - це детальний інструмент для моніторингу споживання енергії на рівні процесів . PowerAPi - це проміжне програмне забезпечення на основі Scala, яке реалізує API для моніторингу програм у режимі реального часу. Воно оцінює споживання енергії різними апаратними компонентами (ПРОЦЕСОР, ОПЕРАТИВНА ПАМ'ЯТЬ, ЖОРСТКИЙ ДИСК тощо). Оцінили його точність порівняно з аналізатором живлення powerspy2 та показали низький середній рівень помилок (тобто 0,5-3%). Слабкістю PowerAPi є те, що він очікує від користувача певного часу для збору вимірювань енергії. Ймовірними найгіршими сценаріями тут є (1) надмірний збір вимірювань (коли програма закінчується, а інструмент все ще вимірює час простою) та (2) неповний збір вимірювань (коли програма все ще працює, але заданий інструментом час роботи занадто короткий).

Варто зазначити, що як Jalen, так і Jolinar використовують PowerAPi як основний компонент для отримання вимірювань енергії. Однак PowerAPi надає дані про вимірювання енергії застосунку на рівні системного процесу, тоді як Jalen та Jolinar співвідносять зібрану інформацію, пов'язану з енергією, з застосунками Java. Порівняно з інструментами, вищезгадані інструменти оцінюють споживання енергії та потужності за допомогою профілювання інструментів, тоді як RAPL використовує лічильники продуктивності. Більше того, як показано в цьому розділі, більшості інструментів бракує сумісності, що може бути обмеженням апаратного або програмного забезпечення.

Віртуальні машини. Joulemeter, інструмент, отримує показники енергоспоживання з ВІРТУАЛЬНИХ МАШИН, серверів, настільних комп'ютерів, ноутбуків та певних програм, відстежуючи використання ресурсів різними апаратними компонентами, такими як ПРОЦЕСОР, ОПЕРАТИВНА ПАМ'ЯТЬ, ЖОРСТКИЙ ДИСК та екран. Його модель оцінює споживання енергії, використовуючи трасування ресурсів ВІРТУАЛЬНОЇ МАШИНИ , яка, у свою чергу, отримує інформацію через лічильники продуктивності. Joulemeter не обмежується вимірюваннями споживання енергії;

Крім того, він пропонує функцію обмеження живлення для кожної віртуальної машини, процедури керування режимом сну та віддаленим пробудженням, що значно зменшує споживання енергії сервером. Однак Joulemeter не підтримується ОС, новішою за Windows 7 .

Представили VMeter, метод моделювання живлення ВІРТУАЛЬНОЇ МАШИНИ для оцінки споживання енергії різними компонентами, такими як процесор, кеш, ДИНАМІЧНА ПАМ'ЯТЬ та ЖОРСТКИЙ ДИСК. Інструмент регулярно контролює використання ресурсів системи та розраховує споживання енергії, використовуючи модель живлення, яка має мінімальні накладні витрати на загальне споживання енергії системою (тобто 0,012%). Для отримання інформації про використання ресурсів з віртуальної МАШИНИ VMeter використовує
лічильники продуктивності та інструмент моніторингу дисків.

БітВатт – це проміжне програмне рішення, яке розраховує споживання енергії програмою всередині ВІРТУАЛЬНОЇ МАШИНИ за допомогою моделі оцінки енергії. Воно є розширенням інструментарію PowerAPi і призначене для збору вимірювань енергії від сучасних і складних мікропроцесорів, що підтримують багатоядерність, гіперпотоковість, DVFS та динамічний розгін. Сильною особливістю BitWatts є збір та агрегація вимірювань енергії від кількох процесів, розташованих у розподіленому середовищі.

Енергетична модель BitWatts порівнювалася з PowerSp у Вимірювач енергії Bluetooth та RAPL. Як результат, вимірювання BitWatts виявилися в межах 2% медіанного коефіцієнта помилок, що є найнижчим показником порівняно з VMeter та Joulemeter. Також, порівняно з VMeter та Joulemeter, BitWatts можна використовувати для аналізу споживання енергії більш складними середовищами, такими як інфраструктура ІНТЕРНЕТУ РЕЧЕЙ та центри обробки даних.

Смартфони. Вимірювання споживання енергії мобільними пристроями здійснюється різними способами. Представили PowerBooter, онлайн-модель оцінки енергії, яка розраховує споживання енергії, поєднуючи вимірювання датчиків напруги акумулятора та швидкості розряду. Таким чином, PowerBooter оцінює споживання енергії без використання зовнішнього апаратного вимірювача потужності. Поєднуючи PowerBooter та PowerTutor , Автори отримали вимірювання енергії на основі активності різних апаратних компонентів, таких як процесор, РК-ДИСПЛЕЙ/OLED, GPS, Wi-Fi та компоненти стільникової мережі.

Аналогічно розробили Android Energy Profiler (AEP) – інструмент, який - пов’язує використання ресурсів процесу з інформацією про напругу та струм. Інформація про напругу та струм генерується за допомогою вбудованого датчика напруги смартфона та використовується AEP для оцінки споживання енергії. AEP не обмежується вимірюваннями споживання енергії; він також може надавати результати продуктивності під час виконання. Однак, на відміну від PowerTutor, AEP надає вимірювання енергії лише для процесора та основної пам’яті. Недоліком AEP є зниження продуктивності під час виконання приблизно на 25%,
яке виникає внаслідок процесу збору даних.

Ще одним інструментом для отримання вимірювань енергії зі смартфонів є GreenOracle. GreenOralce - це модель оцінки енергії, яка після навчання за допомогою різноманітних програм для Android може отримувати вимірювання енергії для будь-якої мобільної програми. Представлена модель, після навчання, може використовуватися розробниками програм для отримання вимірювань енергії без необхідності використання зовнішнього інструменту. Для оцінки споживання енергії GreenOracle використовує динамічне трасування системних викликів (наприклад, strace) та використання процесора.

Інструмент профілювання енергії для Android (PETCA) – це програмний інструмент для профілювання енергії. Особливістю цього інструменту є дрібнозернисті вимірювання енергії, які отримуються на рівні методу. Крім того, порівняно з попередніми роботами, PETCA не потребує калібрування. Точність було перевірено на 54 застосунках для Android та порівняно з апаратним набором інструментів для вимірювання споживання енергії Monsoon. Результати показують, що вимірювання енергії pETrA мало відрізняються від вимірювань Monsoon.

На відміну від усіх попередніх інструментів, PowerBooter пропонує вимірювання енергії для великої кількості компонентів смартфонів. Однак, на відміну від PowerBooter та PETGA, GreenOralce має високий середній рівень помилок. Порівняно з вищезазначеним, GreenOracle має складну модель оцінки енергії, яка після навчання пропонує вимірювання енергії для різних Android-додатків.

Апаратні аналізатори енергії або датчики також можуть отримувати вимірювання споживання енергії з комп'ютерної системи. Однак, розділення грубозернистих вимірювань енергії на програмні або апаратні компоненти є складним завданням. Апаратні засоби моніторингу енергії зазвичай не є автономними інструментами, які потребують додаткових апаратних компонентів, таких як зовнішній пристрій, для отримання вимірювань потужності або енергії. У цьому підрозділі ми опишемо деякі апаратні засоби моніторингу енергії для робочих станцій, серверів та смартфонів.

Інструменти моніторингу робочих станцій та серверів. Лабораторія енергетичного сліду програмного забезпечення (SEFLab) має на меті фіксувати - вимірювання споживання енергії комп'ютерною системою та зіставляти їх з її апаратними компонентами (наприклад, ПРОЦЕСОРОМ, ОПЕРАТИВНОЮ ПАМ'ЯТТЮ, ЖОРСТКИМ ДИСКОМ). Платформа Atom Low-Energy Aware Platform (LEAP) – це тестовий стенд, здатний вимірювати споживання енергії невеликими сегментами коду в ядрі та просторі користувача. Обидва інструменти використовують пристрій Data AcQuisition (DAQ), зовнішній інструмент профілювання енергії, який отримує грубозернисті вимірювання з комп'ютерної системи. Після отримання споживання енергії програмою обидва інструменти намагаються співвіднести отримані вимірювання з часовими позначками та використанням системних ресурсів. Крім того, SEFLab використовує Joulemeter для порівняння його точності з точністю профілера енергії DAQ . Основна відмінність між цими двома підходами полягає в тому, що Atom LEAP намагається зіставити зібране споживання енергії застосунком за допомогою програмних компонентів, тоді як SEFLab зіставляє їх з різними апаратними компонентами. Однак, для використання одного з вищезазначених підходів потрібен ряд інструментів та налаштувань.

Інструменти моніторингу смартфонів. GreenMiner - це експериментальна платформа, яка отримує дані про споживання енергії смартфоном за допомогою запланованих тестів. Усі тести плануються веб-сервісом, тоді як Raspberry Pi відповідає за запуск тестових скриптів на смартфоні за допомогою інтерфейсу Android Debug Bridge. Після запуску тестів, INA21 9 Чіп використовується для запису вимірювань струму та напруги смартфона. Після цього Arduin пристрій відповідає за отримання, обчислення та зберігання всіх вимірювань енергоспоживання з мікросхеми INA219. Згодом Raspberry Pi збирає вимірювання споживання енергії та метадані з пристрою Arduino та передає їх на сервер, де веб-сервіс агрегує, аналізує та зберігає дані експерименту. Крім того, GreenMiner діє як інструмент безперервної інтеграції для проведення тестів та порівняння споживання енергії різними репозиторіями.

Таблиця 3.2

Вплив рефакторингу на енергоефективність

Методи рефакторингу	Енергія (%)	Наслідки для продуктивності	Тестові випадки
Dead Local Variable (Мертва локальна змінна), Non Short Circuit, Parameter By Value, Repeated Conditionals, Self Assignment Variable	< 1 (сер.)	"Запахи коду" (code smells), пов'язані з енергією, не впливають на час виконання	Додатки авторів
Convert Local Variable to Field (Перетворення змінної на поле), Extract Local Variable, Extract Method, Introduce Indirection, Inline Method, Introduce Parameter Object	(−7.5)–4.54	Відсутня чітка кореляція між продуктивністю часу виконання та споживанням енергії для JVM 6 та 7	Вибрані додатки
Replace Method with Method Object та Encapsulate Collection	−7.91–6.99	—	Зразки коду
Loop Unrolling (Розгортання циклу), Loop Unswitching, Method Inline	6.4–50.21	Спостерігається значне підвищення продуктивності часу виконання	Ігровий рушій Cocos2d
Методи рефакторингу	Енергія (%)	Наслідки для продуктивності	Тестові випадки

Завдяки агрегації та аналізу даних він надає аналітичні дані, які можуть допомогти розробникам визначити зміни споживання енергії в процесі розвитку вихідного коду в певних версіях продукту. Подібно до інструментів моніторингу робочих станцій та серверів, представлених у попередньому абзаці, GreenMiner також вимагає ряду інструментів та налаштувань для оцінки споживання енергії програмами.

3.2 Супровід та оптимізація енергоефективності програмного забезпечення

Технічне обслуговування – це процес покращення або виправлення помилок у програмному забезпеченні після його розгортання. Для етапу технічного обслуговування ми обговорюємо методи та інструменти рефакторингу вихідного коду з метою демонстрації економії енергії. У контексті SDLC для

енергоефективності рефакторинг – це практика, спрямована на оптимізацію споживання енергії програмами шляхом модифікацій на рівні коду без зміни базової структури коду. У цьому напрямку це також використовується для аналізу вихідного коду під час розробки програмного забезпечення для виявлення гарячих точок енергоспоживання або помилок. У цьому розділі ми представляємо методи рефакторингу, що застосовуються після фази розгортання (тобто під час фази обслуговування).

Тут ми розглядаємо емпіричні оцінки методів рефакторингу та шаблонів для оптимізації енергопродуктивності, які фахівці з програмного забезпечення можуть використовувати для зменшення споживання енергії додатками. У таблиці 13 наведено деякі зібрані авторами результати . щодо впливу відповідного методу рефакторингу на енергоспоживання та продуктивність під час виконання, а також їхні тестові випадки.

Запропонували концепцію рефакторингу коду для покращення зрозумілості, зручності підтримки та розширюваності існуючого вихідного коду. З цією метою використовували описані шаблони для проведення емпіричних досліджень, які вивчають, як певні шаблони рефакторингу коду впливають на споживання енергії застосунком. Обидва автори використовували різні вбудовані системи, параметри та низку «патернів» для своїх емпіричних досліджень. Зокрема, оцінили шість широко відомих методів рефакторингу на двох версіях віртуальної машини Jave (JVM) (тобто 6 та 7) для дев'яти застосунків. Оцінили 63 з 68 методів рефакторингу на зразках коду, запропонованих Фаулером та ін.; Сахін та ін. дійшли висновку, що кожен метод рефакторингу може збільшити або зменшити споживання енергії застосунком; окрім Extend Local Variables , який завжди зменшує споживання енергії. Аналогічно, Парк та ін. виявили, що деякі « патерни» коду рефакторингу можуть позитивно або негативно змінити споживання енергії. Зокрема, з отриманих результатів Парк та ін. показали, що 33 методи рефакторингу призводять до економії енергії, тоді як решта 30 -ні. Крім того, автори поділилися тим, що споживання енергії між версіями Java, в
контексті методів рефакторингу, не є послідовним.

«Запахи енергетичного коду» – це термін, який стосується неефективних варіантів реалізації, що збільшують споживання енергії. За допомогою свого експериментального дослідження Морісіо та ін. мали на меті визначити низку запахів коду, які можуть змінювати енергоспоживання, продуктивність виконання або навіть і те, й інше. Отже, вони провели свій експеримент на існуючих запахах коду, знайдених у CppChec та FindBug інструменти. Щоб мінімізувати програмний шум від потоків, які можуть виконуватися паралельно та згодом впливати на вимірювання енергії, автори провели свій експеримент на вбудованій системі. Тестуючи дев'ять різних шаблонів рефакторингу, Морісіо та ін. дійшли висновку, що лише п'ять з них зменшують споживання енергії в середньому менш ніж на 1% (див. таблицю 3.2), тоді як решта збільшують споживання енергії або залишають його незмінним. Тим не менш, автори стверджують, що між запахами енергетичного коду та запахами продуктивності не існує жодної кореляції , окрім одного випадку (тобто взаємного виключення). АБО).

Результати, наведені в Таблиці 3.2, показує триманий як приріст, так і втрату споживання енергії, такі як 1%, -7% до 5% та -8% до 7% відповідно. Ми можемо спостерігати, що результати Сахіна та ін. та Парка та ін., які використовували досить схожі шаблони рефакторингу, не сильно відрізняються, навіть попри те, що вони використовували різні мови для їх розробки (тобто Java та C++ відповідно). Морісіо та ін. також використовували шаблони рефакторингу, однак вони продемонстрували лише незначну економію енергії (менше 1%). Більше того показали, що запахи енергетичного коду не впливають на запахи продуктивності і навпаки. Наведені вище результати свідчать про те, що економія енергії, досягнута за допомогою методів рефакторингу, є дуже низькою. Цей факт підкреслює, що застосовані методи рефакторингу в основному зосереджені на покращенні підтримки коду, розширюваності та зрозумілості. Таким чином, вони не можуть запропонувати жодних суттєвих переваг щодо енергоефективності.

Робота в цій галузі спрямована на забезпечення технічного обслуговування шляхом зменшення споживання енергії за допомогою шаблонів рефакторингу коду, вбудованих в інструменти.

Фреймворк для оптимізації енергоспоживання для Android-додатків, який зосереджений на оптимізації енергоспоживання за допомогою набору стратегій рефакторингу розгорнутого вихідного коду. Запропонований фреймворк приймає вихідний код як вхідні дані та аналізує його, щоб отримати дані, пов'язані з енергоспоживанням, та співвіднести їх з базовими блоками коду. Після цього фреймворк виявляє енергетичні гарячі точки вихідного коду та застосовує, автономно або вручну, стратегії рефакторингу (наприклад, розгортання циклів, відключення циклів, вбудовані методи).

Використання такого інструменту рефакторингу може допомогти користувачеві змінити вихідний код існуючих або застарілих систем, щоб зменшити їхнє енергоспоживання та підвищити продуктивність. Автори оцінили свій інструмент у реальних застосуваннях та отримали економію енергії від 6% до 50%.

У цьому розділі ми представляємо результати нашого експерименту. Для кожної практики ми представляємо результати перевірки гіпотез та інші відповідні висновки. Ми також детально обговорюємо результати для кожної практики.

RQ1. У таблиці 3.3 ми підсумовуємо результати перевірки гіпотез. Знаково-ранговий тест Вілкоксона показує, що застосування практики призводить до значного зниження споживання енергії (Z=-3,915, p-значення=0,00009). Таким чином, ми можемо сміливо відхилити нульову гіпотезу.

Таблиця 3.3

Завдання 1: Результати перевірки гіпотез для RQ1 - споживання енергії

Показник	**До (Before)**	**Після (After)**
Медіана (Median)	47.85	35.82
Середнє значення (Mean)	43.83	35.82
Різниця у % (% Diff.)	-	**-25.1 %**
Z-критерій Вілкоксона	-	-3.915
d Коена (Cohen’s d)	-	-140.206 (великий)
g Хеджеса (Hedges’ g)	-	-134.282 (великий)
A Варги та Делейні	-	0 (великий)

RQ2. Що стосується RQ2, то до та після застосування практики розрахунок коефіцієнта кореляції р за Спірменом дав різні результати для 15 з 21 пари змінних ресурс-енергія на системному рівні та для 88 з 168 пар змінних ресурс-енергія на рівні ресурсів. Таким чином, наша нульова гіпотеза відхиляється. У таблиці 3.4 ми наводимо всі значущі коефіцієнти кореляції на системному рівні. У таблиці 3.5, для стислості, ми наводимо лише ті коефіцієнти рівня ресурсів, які мали значну варіацію (Ар > 0,4) до та після застосування практики.

Таблиця 3.4

Завдання 1: Результати перевірки гіпотез для RQ2 - на рівні системи

Ресурс (u)	**Потужність (P)**	**cor(ur,P)**	**cor(ur′,P′)**
CPUusr (користувацький)	Watts	0.673	0.715
CPUsys (системний)	Watts	0.643	-0.163
CPUidI (простій)	Watts	-0.189	-0.319
CPUwai (очікування I/O)	Watts	-0.691	-0.632
DSKread (читання з диска)	Watts	-0.489	0.04
IOread (операції вводу/виводу)	Watts	-0.696	-0.688
MEMbuff (буфер пам'яті)	Watts	0.045	0.127

Наша перевірка гіпотез підтверджує, що наша перша практика, « Використання ефективних запитів», успішно підвищує енергоефективність. Однак необхідно врахувати деякі додаткові фактори. Як видно, зниження споживання енергії у відсотковому вираженні значно менше, ніж зменшення споживання енергії. Це вказує на те, що після застосування практики також спостерігається покращення продуктивності, що є обґрунтованим через відсутність речення ORDER BY . Дійсно, ми повідомляємо про значну різницю в часі виконання: до практики ми виміряли в середньому 257 секунд на запит, тоді як після практики середній час на запит становив 200 секунд.

Зменшення споживання енергії також вказує на різне використання ресурсів, як видно з результатів RQ2: після застосування практики ми можемо спостерігати пряму кореляцію між активністю процесора та споживанням материнської плати/диска, що вказує на більш пропорційну енергоспоживанню поведінку [34]. Ця поведінка стає очевидною при аналізі зв'язку між активністю

процесора та споживанням енергії на рівні системи, як показано на рисунку 3.1. Ще один цікавий висновок стосується зв'язку між операціями вводу/виводу та споживанням енергії: до практики суттєвої кореляції не було, тоді як після ми спостерігаємо негативні коефіцієнти кореляції.

Таблиця 3.5

Завдання 1: Найбільш значущі результати перевірки гіпотез для RQ2 на рівні ресурсів

Ресурс (u)	**Лінія живлення (Pr)**	**cor(ur,Pr)**	**cor(ur′,Pr′)**
CPUusr	MB.5V..Watts (Материнська плата 5В)	0.043	0.583
CPUsys	HDD1.5V..Watts (Диск 1.5В)	-0.553	0.156
IOread	HDD1.5V..Watts (Диск 1.5В)	0.618	0.177
IOread	MEM.12v..Watts (Пам'ять 12В)	-0.076	-0.628
IOread	MB.5V..Watts (Материнська плата 5В)	-0.07	-0.659

Це може бути пов'язано з тим, що операції вводу/виводу зазвичай менш енергоємні, оскільки найбільш енергоємний ресурс, процесор, неактивний.

RQ1. У таблиці 3.5 ми підсумовуємо результати перевірки гіпотез. Знаково-ранговий тест Вілкоксона показує, що застосування практики призводить до значного зниження споживання енергії (Z=-3,929, p-значення=0,00008). Таким чином, ми можемо сміливо відхилити нульову гіпотезу.

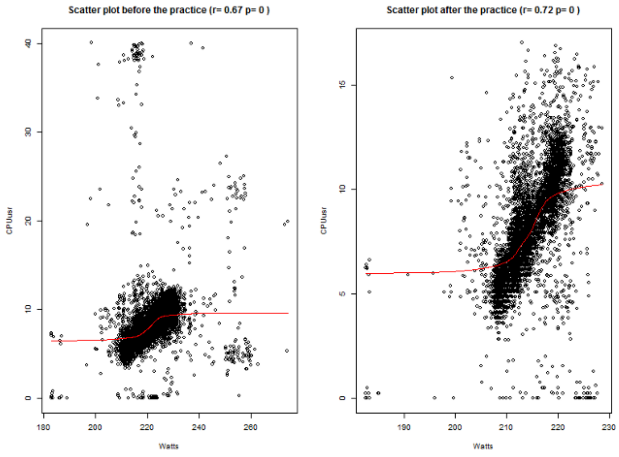

Рисунок 3.1 - Діаграми розсіювання змінних CPUusr і Watts до та після застосування практики 1.

RQ2. Що стосується RQ2, то до та після застосування практики розрахунок коефіцієнта кореляції p за Спірменом дав різні результати для 7 з 21 пари змінних ресурс-енергія на системному рівні та для 55 з 168 пар змінних ресурс-енергія на рівні ресурсів. Таким чином, наша нульова гіпотеза відхиляється. У таблиці 3.6 ми наводимо всі значущі коефіцієнти кореляції на системному рівні. У таблиці 3.6, для стислості, ми наводимо лише коефіцієнти рівня ресурсів, які мали значну варіацію (Ap > 0,4) до та після застосування практики.

Результати перевірки гіпотез, як і для Практики 1, підтверджують корисність Практики 2 для підвищення енергоефективності. З огляду на це, ці дві практики по-різному впливають на споживання енергії та використання ресурсів.

Перш за все, для практики 2 майже немає різниці у впливі між споживанням енергії та потужності, як показано в таблиці 14. Це вказує на менш очевидний вплив на продуктивність: справді, тести до практики показували середній час на запит 0,210 мілісекунди, на відміну від 0,196 мілісекунди після практики, отже, покращення лише на 6%. Однак середнє споживання енергії на запит становить 16,89 пВт·год до практиці та 14,41 пВт·год після цього, зі зниженням на 14%. Таким чином, економія енергії зумовлена не лише скороченням часу виконання.

Таблиця 3.6

Завдання 2: Результати перевірки гіпотез для RQ2 — системний рівень

Ресурс (u)	**Лінія живлення (Pr)**	**cor(ur,Pr)**	**cor(ur′,Pr′)**
CPUusr	MB.5V..Watts (Материнська плата 5В)	0.043	0.583
CPUsys	HDD1.5V..Watts (Диск 1.5В)	-0.553	0.156
IOread	HDD1.5V..Watts (Диск 1.5В)	0.618	0.177
IOread	MEM.12v..Watts (Пам'ять 12В)	-0.076	-0.628
IOread	MB.5V..Watts (Материнська плата 5В)	-0.07	-0.659

Кореляційний аналіз чітко показує, що, як і у випадку з Практикою 1, застосування цієї практики призводить до більш пропорційної енергії поведінки,

оскільки всі коефіцієнти залежності процесора від потужності збільшуються (CPUidl представляє час, витрачений процесором у режимі очікування, що, як і очікувалося, негативно корелює з потужністю). Це явище стає очевидним, якщо подивитися на діаграму розсіювання на рисунку 3.2.

Однак, як показано на блоковій діаграмі на рисунку 4, пам'ять (MEM-12V) також відіграє важливу роль: середнє споживання енергії пам'яттю становить 5,297 Вт·год за один цикл, на відміну від 5,47 та 5,58 Вт·год, споживаних двома процесорами (CPU1 та CPU2 відповідно).

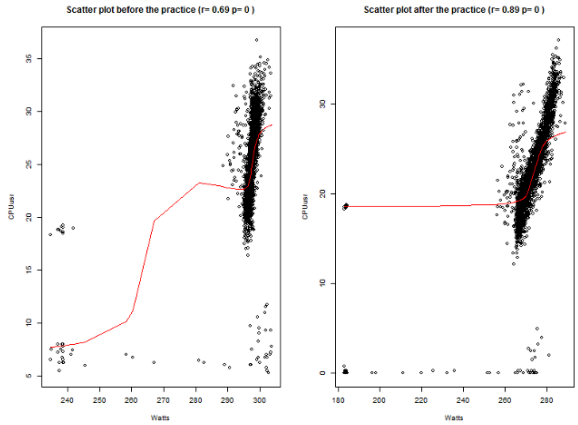

Рисунок 3.2 - Діаграми розсіювання змінних CPUusr і Watts до та після застосування практики 2.

Практики «зеленого» програмного забезпечення. Ми повідомили про дуже значне скорочення споживання енергії, до 25%, що демонструє актуальність досліджень «зеленого» програмного забезпечення. Ця актуальність ще більше зростає, якщо врахувати нові контексти, такі як хмарні обчислення та великі дані, де програмні додатки працюють у незліченній кількості випадків, отже, їхня енергоефективність стає вирішальною. Іншим важливим контекстом є високопродуктивні обчислення: оскільки ми наближаємося до так званої ери екзафлопсних обчислень [39], коли споживання енергії системами високопродуктивних обчислень (HPC) прогнозується на рівні десятків МВт, інженери-розробники апаратного забезпечення та дослідники вже розробляють надзвичайно енергоефективні технології. Програмна інженерія, у свою чергу, ще

не забезпечує узгодженого корпусу знань про енергоефективність.

Дійсно, пов'язана робота, яку ми узагальнили в розділі 2, містить низку висновків у різнорідних контекстах, іноді представлених як пропозиції/керівні принципи для розробників. У цій роботі ми представляємо дві практики «зеленого» програмного забезпечення, задокументовані за допомогою структурованого шаблону, та оцінюємо їхній вплив. Шаблон надає додаткову цінність з точки зору можливості повторного використання практик, тоді як характеристика їхнього впливу допомагає розробникам планувати досягнення певного рівня енергоефективності за умови сумісності практики з областю застосування та іншими можливими обмеженнями.

Дві досліджені нами практики «зеленого» програмного забезпечення можна застосовувати в різних контекстах і програмах. Як ми обговорюємо в наступному розділі, ми не можемо узагальнювати наші висновки на таку широку популяцію. Однак, з відповіді на наше запитання RQ2 видно, що застосування цих практик суттєво впливає на використання системних ресурсів, таких як процесор, пам'ять та введення/виведення, які є досить послідовними, принаймні з логічної точки зору, на всіх IT-платформах. Отже, підвищена пропорційність енергії, що виникає внаслідок застосування цих практик, найімовірніше, спостерігатиметься і в інших подібних контекстах.

Вплив цих практик також дуже різний за своєю природою. Вплив практики 1 на енергоспоживання можна значною мірою пояснити підвищенням продуктивності завдяки меншій кількості операцій вводу/виводу. З іншого боку, вплив практики 2 пояснити складніше: безумовно, використання інструкцій sleep змінює схему виконання різних потоків веб-сервера Apache. Використання інструкції sleep змушує потік звільняти процесор, що дозволяє ефективніше чергувати потоки пулу веб-серверів. Це призводить до порівнянної продуктивності, але ефективнішого використання процесора, а отже, і меншого споживання енергії. Ця складна взаємодія показує, наскільки контекстуалізовані емпіричні дані є критично важливими для фахівців, яким потрібно приймати обґрунтовані рішення щодо енергоефективності своїх програм.

Якщо ми розглянемо вплив «зеленого» програмного забезпечення на процес розробки програмного забезпечення, то побачимо, що енергоефективність представляє нову проблему, яка потенційно впливає на всі фази розробки програмного забезпечення. З архітектурної точки зору, ми вже надали елементи знань про енергоефективність програмного забезпечення, які можна повторно використовувати, у формі архітектурних тактик [40]. Відповідно, практики «зеленого» програмного забезпечення, які можна повторно використовувати, допомагають архітекторам та розробникам програмного забезпечення приймати рішення щодо «зеленого» проектування та вибору впровадження, тим самим забезпечуючи енергоефективність програмного забезпечення. Однак, слід зазначити, що енергоефективність, можливо, є компромісом з іншими атрибутами якості програмного забезпечення. Наприклад, застосування практики 2 («переведення програми в режим сну») може призвести до втрати продуктивності, хоча в нашому експерименті це не було так. Інші практики, які вимагають більш інвазивного рефакторингу програми (наприклад, «зменшення шарів абстракції»), можуть негативно вплинути на зручність обслуговування. Однак, для оцінки таких компромісів потрібні додаткові дослідження, найімовірніше, на рівні архітектур програмного забезпечення. Це частина нашої дослідницької програми [40].

Зрештою, наш експериментальний дизайн та умови є важливою частиною нашого наукового внеску. Спеціалізоване лабораторне середовище для оцінки - енергоефективності програмного забезпечення, таке як SEFLab, є відправною точкою для розробки надійної методології дослідження «зеленого» програмного забезпечення, а також майбутнім тестовим стендом, коли тестування програмного забезпечення на енергоефективність стане звичайною практикою, як це зараз є для інших якостей програмного забезпечення. Інструменти та методи аналізу, які ми застосували, вписуються в більш загальну, багаторазову структуру для «зеленого» програмного забезпечення.

3.3 Оцінка результатів та загрози валідності дослідження

Загалом, наш аналіз показує, що методи та інструменти для забезпечення енергоефективності існують для кожного етапу розвитку СИСТЕМНОГО РОЗВИТКУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ (SDLC). Однак, для того, щоб фахівці з програмного забезпечення могли впровадити та використовувати існуючі інструменти та методи, вирішальними факторами є сумісність, зручність використання та належна підтримка. З цією метою ми вказуємо на низку можливих дослідницьких проблем, які ми визначили в нашому дослідженні. У наступних параграфах ми аналізуємо кожну з цих дослідницьких проблем та пропонуємо напрямки майбутніх досліджень.

У цьому розділі ми представляємо загрози валідності та способи їх пом'якшення. Наша мета - проілюструвати передумови та припущення, що лежать в основі нашого експерименту

Загрози валідності висновків впливають на статистичну значущість результатів. У нашому експерименті ми визначили такі загрози валідності висновків:

Надійність вимірювань. Під час проведення аналізу споживання енергії - точність і правильність вимірювального обладнання мають першорядне значення. З цієї причини ми виконали наші вимірювання в SEFLab, сучасній лабораторії, спеціально побудованій для проведення аналізу споживання енергії. Ми також співпрацювали зі співробітниками та техніками, які створили лабораторію, щоб забезпечити найвищу якість вимірювань.

Як згадувалося, застосування практик «зеленого» програмного забезпечення до наших учасників дослідження є складним процесом, який неможливо стандартизувати або автоматизувати (це також є загрозою для валідності конструкції, див. нижче). Отже, ми не можемо гарантувати, що інше впровадження дасть подібні результати. Щоб зменшити цю загрозу, впровадження практики було здійснено двома різними дослідниками, а її значущість було перевірено з експертами в цій галузі.

Загрози внутрішній валідності впливають на інтерпретацію наших висновків щодо причинно-наслідкового зв'язку між лікуванням та результатом.

Призначення методу лікування. Як ми зазначали, наше емпіричне дослідження є квазіекспериментом з міркувань доцільності: призначення практики одному програмному застосунку – це операція, яку наразі неможливо автоматизувати чи рандомізувати. Ми усвідомлюємо, що це заважає нам повністю встановити причинно-наслідковий зв'язок. Частина наших майбутніх зусиль буде присвячена узагальненню практик, щоб зробити їх автоматично застосовними до кількох продуктів, що дозволить рандомізоване призначення.

Інструментація коду. Через використання IEC API, нам довелося додати кілька додаткових викликів до тем нашої програми. Це могло призвести до фактора, що спотворює ситуацію. Щоб зменшити цю загрозу, ми також провели бенчмарк кожної програми перед виконанням інструментації (так званий ванільний сценарій). Це дозволило нам оцінити вплив інструментації та врахувати його.

Загрози валідності конструктів впливають на зв'язок між теорією та спостереженням. Наша головна загроза валідності конструктів стосується операційного пояснення конструктів, тобто практики не формалізовані стандартним та об'єктивним чином, тому їх переклад в операційні конструкції - підлягає інтерпретації. Щоб пом'якшити цю загрозу, ми задокументували практики наскільки нам відомо, та надали посилання на їхні джерела. Однак ми прагнемо, щоб дослідники оскаржили нашу інтерпретацію та надали додаткові приклади для покращення нашої бази знань про практики «зеленого» програмного забезпечення.

Загрози зовнішній валідності впливають на узагальнення наших висновків. Цей аспект також обговорюється, де ми описуємо наше дослідницьке бачення. Ми визначили такі загрози зовнішній валідності:

Вибір суб'єктів дослідження. З міркувань доцільності, а також через складність застосування практики, ми розробили дослідження з одним об'єктом, тому для нього обрали лише два програмні додатки. Відповідно, ми не можемо гарантувати, що наші суб'єкти дослідження є репрезентативними для всієї популяції. Щоб зменшити цей ризик, ми обрали суб'єкти дослідження якомога

репрезентативними, тобто широко використовувані програмні додатки з відкритим кодом.

Експериментальні умови. Ми проводили наші експерименти в контрольованому середовищі. Тому ми не можемо гарантувати, що наші результати будуть такими ж в інших умовах, наприклад, у виробничому середовищі компанії. Однак, тестова машина та версії застосунку були максимально сучасними та широко використовуються в промислових умовах.

RC1. Вибір конфігурацій та параметрів.

Паралельне програмування доведено як корисне щодо споживання енергії шляхом застосування відповідних конфігурацій та параметрів, таких як кількість потоків, розмір даних та локальність даних. Як зазначили Камбадур та Кім, можна досягти економії енергії на 55% та продуктивності виконання на 69%, якщо конфігурації та параметри програми налаштовані ефективно. Аналогічно, як показано в розділі 4 , використання наближених методів може призвести до зниження енергоспоживання на 10-50% для застосувань, стійких до неточностей. Однак відкритою проблемою як для паралельних, так і для наближених обчислень є (i) пошук частин вихідного коду, які можна оптимізувати, та (ii) вибір відповідних параметрів і конфігурацій для зменшення споживання енергії. Можливим підходом може бути аналізатор вихідного коду, який може позначати можливі точки підвищеного енергоспоживання та пропонувати оптимізацію енергоспоживання, таку як вибір достатньої кількості потоків або визначення частин, які можуть отримати користь від наближених обчислень.

RC2. Обмежене дослідження різних мов програмування.

Один і той самий додаток, розроблений різними мовами програмування, відрізняється за споживанням енергії, часом виконання та продуктивністю. Мови програмування, такі як C та C++, можуть бути складними з точки зору безпеки та надійності управління пам'яттю; проте вони окупаються розробникам меншим споживанням енергії та, водночас, кращою продуктивністю під час виконання. Тим не менш, дослідники дослідили лише невелику частину доступних мов програмування. Інші можуть бути корисними для програм, що працюють на різних

комп'ютерних системах та в різних областях, таких як (ВИСОКОПРОДУКТИВНІ ОБЧИСЛЕННЯ та ІНТЕРНЕТ РЕЧЕЙ).

RC3. Вибір відповідної структури даних.

Як представлено, вибір енергоефективних структур даних має вирішальне значення, оскільки він може суттєво вплинути на розсіювання енергії застосунком. У деяких випадках вибір найефективнішої структури даних у реальних застосунках, таких як Google Gson, Xalan, Tomcat та Huffman Encoder, призвів до економії енергії на 38%. Однак, знати, коли вибирати певні структури даних без необхідності експериментувати, є досить амбітним та складним завданням. Існуючі інструменти, такі як SEEDS, можуть пропонувати структури даних для Java-застосунків; але SEEDS обмежується вибором інтерфейсу колекції черг і ще не доступний для публічного використання. Можливим напрямком дослідження тут є вивчення більшої кількості типів структур даних, визначення тих, які є більш енергоефективними для вибраних випадків, та оформлення цих знань у вигляді інструменту рефакторингу, який може запропонувати заміну реалізацій колекцій.

RC4. Взаємодія, зручність використання та точність для підтримки інструментів.

У розділі ми виявили, що більшість інструментів моніторингу енергоспоживання орієнтовані на певні архітектури ПРОЦЕСОРІВ або операційних систем, що ускладнює їх використання в різних обчислювальних середовищах.

Крім того, налаштування цих інструментів часто потребує багато часу та зусиль. Наприклад, для точного отримання вимірювань енергії користувач повинен знати про такі конфігурації, як напруга процесора на певній частоті. Ще одним складним завданням є оцінка програмних інструментів моніторингу енергії. Для цього багато дослідників порівнюють вимірювання енергії, отримані з апаратних аналізаторів потужності, з результатами запропонованих ними інструментів. Однак важко порівняти або співвіднести ці вимірювання, оскільки багато апаратних аналізаторів потужності мають низьку частоту дискретизації та збирають вимірювання енергії грубозернистим способом. Для цього точні та універсальні

програмні інструменти моніторингу енергії мають першорядне значення. Це сприятиме ширшому впровадженню розробниками програмного забезпечення.

У майбутній роботі також слід враховувати еволюцію процесів розробки програмного забезпечення шляхом розробки інструментів, що підходять для гнучкої розробки програмного забезпечення. У цьому напрямку енергоефективні підходи до розробки програмного забезпечення повинні бути тісно інтегровані з інструментами моніторингу енергії, які постійно оцінюють реальне споживання енергії на апаратному рівні та надають зворотний зв'язок процесу розробки, що може бути використано для точного налаштування вихідного коду для економії енергії. Такі знання є критично важливими та обов'язковими, щоб дозволити фахівцям з розробки програмного забезпечення точно визначати приріст енергії у дедалі складніших комп'ютерних системах та областях, таких як хмарні середовища, центри обробки даних та великі інфраструктури інтернету речей.

3.4 Висновки по розділу

У третьому розділі було розглянуто практичну реалізацію та оцінку енергоефективності програмних рішень. Проаналізовано інструменти вимірювання та тестування енергоспоживання програмного забезпечення, що дозволяє об’єктивно оцінювати його ефективність. Досліджено процес супроводу та оптимізації енергоефективності програмних систем у реальних умовах експлуатації. Також проведено оцінку отриманих результатів із врахуванням можливих загроз валідності дослідження. Підсумкові результати підтверджують доцільність використання методів енергоефективної розробки для зменшення ресурсних витрат і підвищення ефективності програмного забезпечення.

ВИСНОВКИ

У ході виконання роботи було досліджено та частково реалізовано підходи до розробки енергоефективного програмного забезпечення, що є комплексним процесом, спрямованим на зменшення енергоспоживання програмних систем при збереженні їх продуктивності, надійності та функціональності. У роботі проаналізовано сучасні підходи до забезпечення енергоефективності, визначено ключові фактори впливу на енергоспоживання, а також розглянуто інструменти та методи оптимізації програмних рішень. Проведене дослідження охоплює етапи аналізу вимог, проектування архітектури, реалізації, тестування та оцінки ефективності програмного забезпечення.

На початковому етапі було виконано аналіз предметної області, що дозволило визначити основні поняття, принципи та вимоги до енергоефективного програмного забезпечення. Особливу увагу приділено виявленню проблем, пов’язаних із надмірним споживанням ресурсів, неефективними алгоритмами та неоптимальними архітектурними рішеннями. Визначено, що досягнення енергоефективності можливе лише за умови комплексного підходу, який охоплює всі етапи життєвого циклу програмного продукту.

У процесі дослідження методів та інструментів розробки було розглянуто архітектурні та дизайнерські підходи, що сприяють зниженню енергоспоживання, зокрема оптимізацію алгоритмів, ефективне управління ресурсами та використання паралельного програмування. Встановлено, що правильний вибір архітектури та методів виконання програм безпосередньо впливає на рівень споживання енергії та загальну продуктивність системи.

Реалізація та оцінка ефективності рішень включали аналіз сучасних інструментів для вимірювання енергоспоживання програмного забезпечення, що дозволило отримати об’єктивні дані щодо ефективності застосованих підходів. Проведено оцінку впливу різних оптимізацій на продуктивність і енергоспоживання, а також розглянуто питання супроводу програмного

забезпечення з урахуванням необхідності підтримки енергоефективності протягом усього життєвого циклу. Окремо проаналізовано можливі загрози валідності отриманих результатів, що дозволило підвищити достовірність дослідження.

Загалом результати роботи підтверджують, що впровадження методів енергоефективної розробки дозволяє суттєво знизити споживання обчислювальних ресурсів і електроенергії, підвищити продуктивність програмних систем та зменшити витрати на їх експлуатацію. До переваг запропонованого підходу можна віднести універсальність застосування, можливість інтеграції з сучасними технологіями та позитивний вплив на екологічні аспекти використання ІТ.

Разом із тим існують певні обмеження, зокрема необхідність додаткових витрат часу на оптимізацію, складність точного вимірювання енергоспоживання та залежність результатів від апаратного середовища. У подальшому доцільно розширити дослідження шляхом інтеграції енергоефективних підходів із хмарними технологіями, використанням штучного інтелекту для автоматичної оптимізації, а також розробкою універсальних стандартів оцінки енергоефективності програмного забезпечення.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Green Software Foundation. (2024). Software Carbon Intensity Specification. https://greensoftware.foundation/projects/software-carbon-intensity-specification

2. Microsoft. (2025). .NET Performance and Energy Efficiency. https://learn.microsoft.com/en-us/dotnet/core/performance/

3. Energy Efficient Computing. (2023). IEEE Standards Overview. https://standards.ieee.org/

4. Intel. (2024). Energy-Efficient Performance Optimization. https://www.intel.com/content/www/us/en/developer/topic-technology/performance/

5. Google. (2024). Energy Efficient Computing for Data Centers. https://cloud.google.com/sustainability

6. Koomey, J. (2011). Growth in Data Center Electricity Use. Analytics Press.

7. Hindle, A. (2015). Green Mining: Software Energy Consumption. IEEE Software. https://doi.org/10.1109/MS.2015.65

8. Pathak, A., Hu, Y. C., & Zhang, M. (2012). Where is the energy spent inside my app? ACM. https://doi.org/10.1145/2348543.2348550

9. Li, D., & Halfond, W. (2014). An Investigation into Energy-Saving Programming Practices. IEEE. https://doi.org/10.1109/ICSE.2014.691

10. Pereira, R. et al. (2017). Energy Efficiency in Software Engineering. Springer. https://doi.org/10.1007/978-3-319-53745-2

11. Mahmoud, S., Ahmad, I. (2013). A Green Model for Sustainable Software Engineering. Journal of Software.

12. IBM. (2024). Sustainable Software Engineering Practices. https://www.ibm.com/cloud/learn/sustainable-software

13. Fowler, M. (2002). Patterns of Enterprise Application Architecture. https://martinfowler.com/eaaCatalog/

14. Martin, R. C. (2017). Clean Architecture. Pearson.

15. Gamma, E. et al. (1994). Design Patterns. Pearson.

16. Freeman, E., Robson, E. (2021). Head First Design Patterns. O'Reilly.

17. Microsoft Azure. (2025). Sustainability in Cloud Computing. https://learn.microsoft.com/en-us/azure/sustainability/

18. AWS. (2025). Sustainability Pillar – Well-Architected Framework. https://docs.aws.amazon.com/wellarchitected/latest/sustainability-pillar/

19. Google Cloud. (2025). Carbon-Aware Computing. https://cloud.google.com/sustainability

20. MongoDB. (2025). Performance Optimization Guide. https://www.mongodb.com/docs/manual/administration/analyzing-mongodb-performance/

21. PostgreSQL. (2024). Performance Tuning. https://www.postgresql.org/docs/current/performance-tips.html

22. .NET Best Practices. (2023). https://learn.microsoft.com/en-us/dotnet/core/extensions/best-practices

23. Visual Studio Profiler. (2025). https://learn.microsoft.com/en-us/visualstudio/profiling/

24. JetBrains. (2024). dotTrace Performance Profiler. https://www.jetbrains.com/profiler/

25. Linux Foundation. (2024). Power Management in Linux. https://www.kernel.org/doc/html/latest/power/

26. Android Developers. (2025). Optimize Battery Life. https://developer.android.com/topic/performance/power

27. Apple. (2025). Energy Efficiency Guide for iOS Apps. https://developer.apple.com/documentation/xcode/energy_efficiency

28. OWASP. (2024). Secure Coding Practices. https://owasp.org/www-project-top-ten/

29. Cypress. (2025). Testing Documentation. https://docs.cypress.io/

30. Selenium. (2025). Documentation. https://www.selenium.dev/documentation/

31. Postman. (2025). API Testing Guide. https://learning.postman.com/docs/

32. Docker. (2025). Resource Constraints. https://docs.docker.com/config/containers/resource_constraints/

33. Kubernetes. (2025). Resource Management. https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

34. ISO/IEC 25010. (2011). Systems and Software Quality Models. https://www.iso.org/standard/35733.html

35. SPECpower Benchmark. (2024). Energy Benchmarking. https://www.spec.org/power_ssj2008/

36. Energy Star. (2024). Data Center Efficiency Metrics. https://www.energystar.gov/products/data_center_equipment

37. Deloitte. (2025). Green IT and Sustainable Software. https://www2.deloitte.com/

38. Gartner. (2024). Sustainable Software Engineering Trends. https://www.gartner.com/en

39. Kumar, S., Sharma, R. (2023). Energy Efficient Software Systems. Journal of Software Engineering. https://doi.org/10.1186/s40411-023-00189-9

40. Wang, L., Zhang, Y. (2025). Energy-Aware Software Design. ACM Transactions on Software Engineering. https://doi.org/10.1145/3647890

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: "Розробка енергоефективного програмного забезпечення"

Обсяг пояснювальної записки: 69 аркушів

Дата закінчення бакалаврської роботи 10 червня 2026р.

Підпис студента _____