ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Алгоритм інтелектуального енергозбереження для ноутбуків з операційною системою Windows на основі аналізу поведінки користувача»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Віталій КИСЮК
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-62
Віталій КИСЮК

Керівник: Юрій ЗАДОНЦЕВ
канд.техн.наук

Рецензент: _____
Ім'я, ПРІЗВИЩЕ

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кисюку Віталію Вікторовичу

1. Тема кваліфікаційної роботи: «Алгоритм інтелектуального енергозбереження для ноутбуків з операційною системою Windows на основі аналізу поведінки користувача»

керівник кваліфікаційної роботи Юрій ЗАДОНЦЕВ, канд. техн. наук,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467.

2. Строк подання кваліфікаційної роботи «19» грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література з питань енергозбереження та оцінки стану акумуляторів, вимірювальні дані (напруга батареї, статус живлення), технічна документація Windows Power Management (ACPI, Battery API), технічні характеристики тестових ноутбуків, результати вимірювань при експериментах та вимоги до точності оцінки SoC.

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

1. Аналіз існуючих підходів до оцінки заряду акумулятора (SoC) та механізмів енергозбереження в ОС Windows.
2. Розробка математичної моделі оцінки SoC на основі вимірюваної напруги та її калібрування.
3. Розробка модифікованого методу адаптивного енергозбереження: логіка прийняття рішень, корекції напруги та набір дій (Wi-Fi, Bluetooth, GPU, служби, яскравість, TurboBoost).
4. Реалізація програмного модуля (EcoControl): інтерфейс, модулі збору даних, модуль обчислення SoC, модуль керування параметрами системи.
5. Проведення експериментальної валідації та порівняльного аналізу ефективності (зменшення енергоспоживання, час автономної роботи, точність оцінки SoC).
6. Висновки, рекомендації щодо впровадження та подальших досліджень.

5. Перелік ілюстративного матеріалу: *презентація*
    1. Математична модель оцінки реального заряду акумулятора (SoC), формула та графік залежності U → SoC (лінійне наближення та приклади корекції).
    2. Блок-схема алгоритму (flowchart) прийняття рішень.
    3. Діаграма послідовності роботи модуля енергозбереження.
    4. Результати моделювання/вимірювань: графіки часу автономної роботи, енергоспоживання (Windows vs EcoControl).
    5. Таблиці порівняльних показників (MAE SoC, збережене енергоспоживання, розмір застосунку).
    6. Демонстрація роботи модуля на прикладі реального ноутбука (скрінкасти/log-файли).

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	31.10.2025-04.11.2025	
2	Вивчення методів вимірювання напруги та специфікацій Windows Power Management (ACPI, API)	05.11.2025-09.11.2025	
3	Дослідження існуючих методів оцінки SoC та підходів до автоматизації енергозбереження	10.11.2025-14.11.2025	
4	Розробка математичної моделі оцінки SoC на основі напруги та методів калібрування	15.11.2025-19.11.2025	

5	Розробка модифікованого методу адаптивного енергозбереження (логіка, правила, чекбокси)	20.11.2025-21.11.2025	
6	Реалізація програмного модуля EcoControl (UI, збір даних, SoC-модуль, керування компонентами)	22.11.2025-26.11.2025	
7	Оформлення роботи: вступ, висновки, реферат	27.11.2025-29.11.2025	
8	Розробка демонстраційних матеріалів	30.11.2025-19.12.2025	

Здобувач вищої освіти _____ Віталій КИСЮК
(підпис)

Керівник
кваліфікаційної роботи _____ Юрій ЗАДОНЦЕВ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 71 стор., 5 табл., 6 рис., 25 джерел.

Мета роботи – збільшення часу роботи ноутбука від акумулятора з операційною системою Windows за рахунок застосування алгоритму енергозбереження на основі аналізу поведінки користувача.

Об'єкт дослідження – процеси енергозбереження у ноутбуках.

Предмет дослідження – методи та алгоритми енергозбереження у ноутбуках з операційною системою Windows.

У роботі використано різноманітні методи, такі як опитування системних інтерфейсів (WMI, WinAPI), обробка подій живлення (WM_POWERBROADCAST, SystemEvents), лінійна апроксимація OCV→SoC, EWMA-згладжування, емпіричне калібрування меж Umin/Umax, а також методи експериментальної валідації та порівняльного аналізу. Для реалізації застосовано об'єктно-орієнтоване програмування (C# / WinForms) та інструменти логування результатів.

Проведено аналіз сучасних методів оцінки заряду та механізмів енергозбереження (стандартні Power Plans Windows, Battery Saver, популярні утиліти на кшталт BatteryCare). Встановлено їхні обмеження: залежність від даних контролера батареї (BMS), відсутність корекції під поточне навантаження та обмежені засоби автоматичного керування апаратними компонентами.

Розроблено та оптимізовано математичну модель оцінки стану заряду (SoC) на основі напруги з урахуванням поправок на навантаження (ΔU), а також модифікований метод адаптивного енергозбереження з логікою прийняття рішень за порогами і набором дій (Wi-Fi, Bluetooth, дискретна GPU, служби, TurboBoost, яскравість, очищення RAM).

Реалізовано програмний модуль **EcoControl** з інтерфейсом користувача, модулем збору даних, SoC-модулем та механізмом виконання дій згідно з обраними чекбоксами.

Проведено експерименти для валідації розробленого методу у трьох сценаріях (офісна робота, відтворення відео, пікове навантаження). Результати демонструють зменшення середнього енергоспоживання на **5–20 Вт**, збільшення часу автономної роботи на **1.5–4 години** та оцінку точності SoC близько **~90%** у порівнянні зі стандартними режимами Windows. Також відзначено, що EcoControl забезпечує ширшу автоматизацію дій і має мінімальний розмір виконуваного файлу (~83 КБ).

Результати дослідження підтверджують ефективність запропонованого підходу та перспективність його впровадження для підвищення автономності ноутбуків. Подальші дослідження можуть бути спрямовані на вдосконалення калібрування SoC, інтеграцію з додатковими апаратними інтерфейсами та розширення набору енергозберігаючих стратегій.

КЛЮЧОВІ СЛОВА: ECOCONTROL, SOC, ОЦІНКА ЗАРЯДУ, ЕНЕРГОЗБЕРЕЖЕННЯ, WINDOWS POWER MANAGEMENT, АВТОМАТИЗАЦІЯ КОМПОНЕНТІВ, ECOQOS, КОРЕКЦІЯ НАПРУГИ.

ABSTRACT

Text part of the master's qualification work: 71 pages, 6 pictures, 5 table, 25 sources.

The purpose of the work – to increase the battery life of a laptop running Windows OS by applying an energy-saving algorithm based on user behavior analysis.

Object of research – energy saving processes in laptops.

Subject of research – methods and algorithms of energy saving for laptops running Windows.

Summary of the work: This work employs a variety of methods, including system interface polling (WMI, WinAPI), handling of power events (WM_POWERBROADCAST, SystemEvents), linear OCV→SoC approximation, EWMA smoothing, empirical calibration of Umin/Umax, and experimental validation and comparative analysis. The software implementation is based on object-oriented programming (C# / WinForms) with logging and modular design.

An analysis of existing charge estimation and energy saving mechanisms (Windows Power Plans, Battery Saver, utilities such as BatteryCare) was carried out, revealing limitations: reliance on battery controller (BMS) reports, lack of correction for current load, and limited capabilities for automated control of hardware components.

A mathematical model for State-of-Charge (SoC) estimation based on battery voltage with load correction (ΔU) was developed and optimized, together with a modified adaptive energy-saving method implementing decision logic by thresholds and a set of actions (Wi-Fi, Bluetooth, discrete GPU, services, TurboBoost, display brightness, RAM cleaning). The EcoControl software module was implemented, including a UI, data acquisition, SoC computation, and an action execution subsystem governed by user-selectable checkboxes.

Experiments were performed for validation in three scenarios (office work, video playback, peak load). Results demonstrate a reduction in average power consumption by 5–20 W, an increase in battery runtime by 1.5–4 hours, and SoC estimation accuracy around ~90% compared to standard Windows modes. The solution also provides broader automation and an extremely small executable footprint (~83 KB).

The findings confirm the effectiveness of the proposed approach and its potential for improving laptop autonomy. Further work may focus on enhanced SoC calibration, integration with additional hardware interfaces, and expanding energy-saving strategies.

KEYWORDS: ECOCONTROL, SOC, BATTERY STATE ESTIMATION, ENERGY SAVING, WINDOWS POWER MANAGEMENT, COMPONENT AUTOMATION, ECOQOS, VOLTAGE CORRECTION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SoC – стан заряду акумулятора у відсотках

Uпоточна – поточна напруга батареї у вольтах

Umin – напруга при безпечному мінімумі (виробнича/калібрована межа)

Umax – напруга при повному заряді (виробнича/калібрована межа)

ΔU – поправка напруги на поточне навантаження (врахування CPU, яскравості, мережі тощо)

MAE - похибка моделі SoC після всіх корекцій (ΔU, формули, згладжування), тобто наскільки формула відхилена в середньому.

BMS – система керування батареєю (Battery Management System) – контролер батареї

ACPI – інтерфейс керування енергоспоживанням (Advanced Configuration and Power Interface)

WMI – інструменти Windows для збору системної інформації (Windows Management Instrumentation)

WinAPI / SystemEvents – програмні інтерфейси Windows для обробки подій (PowerModeChanged, WM_POWERBROADCAST)

EcoQoS – режим ефективності програм (енергетично-орієнтований Quality of Service)

CPU – процесор (центральний процесор)

GPU – графічний процесор (дискретний/інтегрований)

RAM – оперативна пам’ять

UI – графічний інтерфейс користувача (User Interface)

ВСТУП

Особливої актуальності тема набуває в умовах воєнного часу та частих відключень електроенергії, коли ефективне використання заряду акумулятора є критично важливим для забезпечення безперервної роботи ноутбука. Сучасні операційні системи, зокрема Windows, часто покладаються на інформацію, що надходить від контролера батареї (BMS), що у випадку некоректної роботи контролера або після заміни акумуляторних банок призводить до неправильного відображення залишку енергії. Крім того, існуючі стандартні механізми енергозбереження виконують обмежений набір дій і не адаптуються в реальному часі під поточне навантаження системи.

Метою магістерської кваліфікаційної роботи є покращення роботи ноутбуків з операційною системою Windows за рахунок застосування алгоритму інтелектуального енергозбереження, що аналізує поведінку користувача та поточні системні параметри.

Для досягнення поставленої мети були визначені наступні завдання дослідження:

1. Провести огляд та аналіз існуючих підходів до оцінки заряду акумулятора (SoC) та механізмів енергозбереження у Windows.
2. Розробити математичну модель оцінки стану заряду на основі вимірюваної напруги з урахуванням корекцій на навантаження.
3. Запропонувати модифікований адаптивний метод енергозбереження з логікою прийняття рішень та набором дій для керування компонентами системи.
4. Реалізувати програмний модуль EcoControl (UI, модуль збору даних, SoC-модуль, контролер режимів, виконавець дій).
5. Провести експериментальну валідацію розробленого підходу та порівняльний аналіз з існуючими рішеннями.

Об’єктом дослідження є процеси енергозбереження у ноутбуках.

Предметом дослідження є методи та алгоритми енергозбереження у ноутбуках з операційною системою Windows.

У роботі було використано наступні методи: аналіз літературних джерел і технічної документації (ACPI, WMI), моделювання математичних залежностей, емпіричне калібрування параметрів, програмна реалізація на C# (WinForms), експериментальні вимірювання енергоспоживання та порівняльний аналіз результатів.

Наукова новизна одержаних результатів полягає у:

1. Запропоновано математичну модель оцінки SoC за напругою з урахуванням поправок на навантаження, придатну для застосування на ноутбуках із некоректними показниками контролера.
2. Розроблено адаптивний метод автоматичного керування компонентами системи (Wi-Fi, Bluetooth, дискретна GPU, служби, TurboBoost, яскравість), що базується на аналізі поведінки користувача і поточних параметрів системи.
3. Реалізовано практичний модуль EcoControl із мінімальним розміром виконуваного файлу та можливістю гнучкого налаштування поведінки через інтерфейс чекбоксів.

Практичне значення одержаних результатів дослідження можуть бути використані для підвищення тривалості роботи ноутбуків від акумулятора у політичних/екстрених умовах, вбудовані у спеціалізовані утиліти або інтегровані у рішення для дистанційної роботи та мобільних операційних середовищ.

Апробація результатів. Основні положення та результати дослідження були представлені на внутрішніх демонстраціях та тестуванні в лабораторних умовах

1 ТЕОРЕТИЧНІ ОСНОВИ ОЦІНКИ ЗАРЯДУ ТА МЕТОДИ ЕНЕРГОЗБЕРЕЖЕННЯ

У цьому розділі викладено детальні теоретичні відомості, які лягли в основу розробки практичної системи EcoControl. Особливу увагу приділено складнощам оцінки стану заряду акумулятора (SoC), що є багатофакторною задачею, залежною як від фізичних характеристик елементів живлення, так і від умов їх експлуатації. Розглянуто вплив температури, циклів заряд–розряд, старіння батареї та динамічних навантажень на точність визначення залишкового заряду. Окремо проаналізовано основні методи оцінки SoC, серед яких метод відкритої напруги (OCV), метод кулонного підрахунку (coulomb counting), а також комбіновані підходи, що поєднують кілька джерел даних для підвищення точності. Крім того, розглянуто архітектуру та можливості стандартних механізмів енергозбереження у ОС Windows, зокрема Power Plans та Battery Saver, які забезпечують базову адаптацію системи до рівня заряду. Матеріал цього розділу призначений для обґрунтування вибору підходів, формування алгоритмічної бази та методики експериментальної валідації, що надалі використовуються у практичній частині роботи.

1.1 Аналіз проблематики оцінки стану акумулятора і існуючих підходів

До загальних положень та визначень можна віднести, що стан заряду акумулятора (State of Charge, SoC) — це відносне значення, яке характеризує частку заряду, залишеного в батареї, відносно її номінальної ємності. SoC зазвичай виражається у відсотках від 0% (повністю розряджений) до 100% (повністю заряджений). Для систем енергоменеджменту ноутбука SoC є базовою величиною для прийняття рішень: від виведення повідомлень до активації енергозберігаючих політик.

Точна і стабільна оцінка SoC — нетривіальне завдання з огляду на фізичні характеристики акумуляторів (літій-іонні, літій-полімерні тощо), їхній режим експлуатації, старіння та обмеженість апаратних інтерфейсів у споживчих пристроях.

Фізичні властивості акумуляторів, що впливають на SoC, включають:

— Для сучасних літій-іонних елементів крива напруги від SoC має ділянки: початкова швидка зміна при високих зарядних станах, практично пласка ділянка посередині (де напруга майже не змінюється при зміні SoC) та крутий спад при низьких SoC. Це означає, що в проміжних ділянках пряме відображення SoC по напрузі дає низьку чутливість і високу похибку.

— Під час споживання струму відбувається падіння напруги, яка описана формулою 1.1:

$$\Delta V = I \cdot R_{\text{int}}, \tag{1.1}$$

де ΔV – падіння напруги;

I – струм;

R_{int}— внутрішній опір батареї.

При пікових навантаженнях напруга на клемах може бути істотно нижчою за «справжню» OCV, що вводить системи в оману, якщо не робити корекцію.

— Параметри батареї залежать від температури: ємність, внутрішній опір та крива U–SoC змінюються, отже необхідні температурні поправки.

— Зі старінням акумулятора зменшується доступна ємність, змінюються внутрішні параметри; це означає, що одна й та сама напруга може відповідати різним SoC у нової та старої батареї.

— В багатьох батареях є кілька снованих груп/банок; зміни в окремих банках (наприклад, після ремонту) впливають на загальний відгук системи та вимагають перекалібрування.

Практичні проблеми в ноутбуках проявляються у наступному:

— Відсутність відкритого інтерфейсу для даних струму: не всі виробники надають покази струму або токових сенсорів через ACPI/WMI.

— Некоректна робота BMS після ремонту або заміни банок — поширена причина того, що Windows показує 0% або некоректні значення.
— Наявність фонових процесів, короткочасних піків навантаження, що «шумлять» у вимірюванні напруги.
— Неоднорідність апаратних платформ — рішення повинно бути універсальним і не вимагати спеціального обладнання.

Нижче коротко перелічено основні підходи існуючих методів оцінки **SoC** з їхніми перевагами та недоліками.

OCV (Open Circuit Voltage) – це метод по напрузі, це коли по своїй напрузі у стані спокою акумулятор має відповідний SoC.

— Переваги: простота, не потребує струмових датчиків.
— Недоліки: вимагає режиму спокою або компенсації для навантажених умов; чутливість змін напруги слабка в середньому діапазоні; температура/старіння змінюють криву.

Coulomb counting (інтегрування струму) – це інтегрування струму в часі, яка описана формулою 1.2:

$$Q(t) = Q(0) - \int_0^t I(\tau)d\tau, \quad (1.2)$$

Де $Q(t)$— залишкова ємність на момент часу t;

$Q(0)$— початкова ємність;

$I(\tau)$— струм у момент часу τ(позитивний при розряді)

— Переваги: високий динамічний контроль, добро працює при надійних потоках даних струму.
— Недоліки: вимагає точних вимірювань струму, накопичує дрейф, потребує періодичної калібрування.

Комбіновані методи (фільтри, моделі) – це поєднання інтегрування струму з періодичною корекцією по напрузі або застосування фільтрів (Калман, EKF) для фузії джерел даних.

— Переваги: підвищена точність, менше дрейфу.

— Недоліки: складність математичного апарату, потреба даних (струм, температура), обчислювальні витрати.

Модельні – це RC-ланцюги, еквівалентні схеми.

— Переваги: дає можливість врахувати динаміку батареї.

— Недоліки: потребує ідентифікації параметрів.

ML – використовують історичні дані та ознаки (вольтаж, споживання, профілі використання) для побудови моделі SoC/SoH.

— Переваги: мають потенціал.

— Недоліки: вимагають датасетів.

Наприкінці варто зазначити, що для універсального програмного продукту (без додаткового апаратного обладнання) оптимальним є **спрощений комбінований підхід**: лінійна (або кусочно-лінійна) апроксимація U→SoC з емпіричною корекцією напруги під навантаженням (ΔU), калібрування меж $U_{\min}, U_{\max}$ і періодичне підстроювання (калібрування) при можливості. Такий підхід дає прийнятну точність за реалістичних обмежень.

1.2 Огляд стандартних механізмів енергозбереження у Windows

Архітектура енергоменеджменту в Windows реалізує управління енергоспоживанням на декількох рівнях:

— **Firmware/ACPI/BMS** — апаратний шар, який повідомляє ОС про стан батареї, рівень заряду, температуру, стан живлення.

— **Kernel/Drivers** — драйвери, що виконують низькорівневі дії (P-state/C-state для CPU, управління живленням пристроїв).

— **Power Policy Manager** — компонент ОС, що зберігає та застосовує плани живлення (Power Plans), відслідковує події та змінює політику.

— **System Services / OEM Utilities** — служби та додатки, які можуть додатково впливати на енергетику (наприклад, OEM утиліти, драйвери GPU, служби моніторингу).

Цей стек забезпечує узгоджене управління живленням пристрою, але реальна ефективність залежить від якості даних (BMS), драйверів та реалізації OEM.

Power Plans – це конфігурації політик, які визначають поведінку різних підсистем (CPU, диск, дисплей, сплячі режими). Основні плани: **Balanced**, **Power saver**, **High performance**. Кожен план задає набори параметрів (наприклад, мінімальні/максимальні значення P-state, таймаути). Power Plans зручні для користувача, але **не є адаптивними**, так як вони діють за заздалегідь заданими правилами, які не завжди відповідають динаміці навантаження.

Battery Saver – це влаштований режим у Windows, що призначений для швидкого зниження споживання при досягненні певного порога SoC (за замовчуванням ~20%). Основні дії:

— зменшення яскравості дисплея;

— обмеження фонової активності (особливо для Store/UWP додатків);

— використання політик для зниження активності мережі;

— можливе застосування обмежень для процесорів (через Power Plans).

Обмеження Battery Saver:

— Чіткий набір дій, обмежений щодо контролю апаратних компонентів.

— Залежить від коректності SoC від BMS.

— Не вимикає специфічні служби або дискретні пристрої (без додаткових утиліт).

Розробники мають доступ до сигналів, даних, подій живлення та API, які включають:

- **WM_POWERBROADCAST** – повідомлення віконній підсистемі про події живлення; один із кодів – PBT_APMPOWERSTATUSCHANGE (зміна статусу живлення).
- **SystemEvents.PowerModeChanged** – подія .NET для відстеження змін живлення.
- **WMI (Win32_Battery, Win32_PowerPlan, Win32_BatteryStaticData)** – класи для читання інформації про батарею та плани живлення.

- **powercfg** – утиліта командного рядка для налаштування планів живлення.

Виробники ноутбуків постачають власні керівні утиліти (Lenovo Vantage, Dell Power Manager, HP Power Manager), які часто мають доступ до додаткових драйверних інтерфейсів і розширених можливостей. Вони можуть виконувати глибші оптимізації, але, як правило, є прив'язаними до конкретних платформ і мають великий розмір.

Недоліками стандартних механізмів Windows можна віднести:

— **Низький рівень адаптивності** до поточної поведінки користувача.

— **Залежність від BMS**, тобто коли BMS надає хибні дані, Windows приймає неправильні рішення.

— **Обмеженість дій Battery Saver**, тобто коли Battery Saver не охоплює детального апаратного контролю.

— **Відсутність тонкого користувацького контролю** автоматичних сценаріїв (без додаткового ПЗ).

При розробці слід враховувати, що EcoControl повинен доповнювати стандартні механізми Windows шляхом:

— власної оцінки SoC (у випадках, коли дані BMS підозрілі або відсутні);

— додаткової корекції напруги під навантаженням;

— автоматичного керування компонентами, які Windows не контролює (чи контролює не в повному обсязі) — Wi-Fi, Bluetooth, служби, динамічне керування dGPU, TurboBoost, яскравістю, очищення RAM;

— забезпечення гнучкості через чекбокси для користувача.

1.3 Методи оцінки SoC

Метод OCV (оцінка по відкритій напрузі) – це залежність між напругою батареї у відкритому колі (без навантаження) і її зарядом та описана формулою 1.3. Ідеальна реалізація вимагає періоду спокою для відновлення напруги після навантажувальних піків.

Формула (спрощена):

$$SoC \approx f_{\text{OCV}}(U_{\text{OCV}}), \tag{1.3}$$

де f_{OCV} – функція, яка перетворює напругу на відсоток заряду;

U_{OCV} – напруга холостого ходу.

Практичні поліпшення:

- кусочно-лінійні таблиці або інтерполяція;
- калібрування $U_{\min}, U_{\max}$;
- корекція Uна навантаження описана формулою 1.4(у випадку, коли не доступна OCV):

$$U_{\text{еф}} = U_{\text{поточна}} + \Delta U, \tag{1.4}$$

де $U_{\text{еф}}$ – скориговане значення, яке імітує напругу без навантаження (OCV).

$U_{\text{поточна}}$ – фактична напруга батареї під навантаженням, яка зчитується сенсорами;

ΔU — поправка, яка моделює падіння напруги через внутрішній опір та активність системи (CPU, яскравість).

У випадку, коли недоступне значення напруги холостого ходу (OCV), для оцінки заряду використовується скориговане значення напруги. Ефективна напруга $U_{\text{еф}}$визначається як сума поточної виміряної напруги $U_{\text{поточна}}$та поправки ΔU, яка враховує вплив навантаження на систему. Такий підхід дозволяє компенсувати тимчасові просадки напруги, що виникають через внутрішній опір акумулятора та активність компонентів, і забезпечує більш точну оцінку залишкового заряду.

Приклад кусочно-лінійного перетворення:

— $U < 9.8\,V \Rightarrow SoC \approx 0\%$;

— $9.8–11.0\,V \Rightarrow 0–40\%$(лінійно);

— $11.0–12.2\,V \Rightarrow 40–90\%$(плавно);

— $> 12.6\,V \Rightarrow \approx 100\%$.

Coulomb counting (інтегрування струму) реалізує оцінку SoC як результат інтегрування струму від початкового стану та описана формулою 1.5:

$$SoC(t) = SoC(0) - \frac{1}{Q_{nom}} \int_0^t I(\tau)\, d\tau, \qquad (1.5)$$

де Q_{nom} — номінальна ємність (А·год),

I — струм (позитивний при розряді).

Питання практики:

— **Похибка інтегрування** (drift) накопичується і потребує зовнішньої корекції (по напрузі чи при повному циклі заряд-розряд).

— **Точність вимірювання струму**: шунт-резистор або датчик має високий клас точності; у ноутбуках ці дані рідко доступні.

— **Coulombic efficiency**: фактична кількість відданого заряду може відрізнятися від номінальної через відсоток втрат; це треба враховувати.

Переваги для тих систем, де доступний струм:

— швидке сповіщення про зміни заряду при динамічних навантаженнях;

— можливість точного квантування енергії, якщо є точні дані.

Комбіновані підходи — фільтри Калмана, EKF, фузія

Комбіновані підходи (фільтри Калмана, EKF, фузія даних) поєднують інтегрування струму з періодичною корекцією по напрузі або застосовують станні фільтри для фузії вимірювань і моделі батареї.

Ідея: моделювати динаміку батареї (стани – SoC, внутрішній опір, тощо) у вигляді моделі стану, а вимірювання (U, I) використовувати для корекції оцінок. KF дає оптимальну оцінку у випадку лінійної моделі та гаусових шумів; EKF – для нелінійних.

У спрощеному вигляді це реалізується:

— модель стану: $x_{k+1} = Ax_k + Bu_k + w_k$;

— модель вимірювання: $z_k = h(x_k, u_k) + v_k$ (де h може бути нелінійною).

Переваги EKF:

— більш точна оцінка;

— можливість врахувати динаміку внутрішніх процесів батареї.

Недоліки:

— складність налаштування, потреба в точних початкових параметрах;

— обчислювальна складність може бути ресурсозатратно.

Існують інші варіанти, як **Particle Filter, UKF.** Ці методи застосовуються у складніших системах з нелінійними процесами та нечіткими шумами.

Моделі еквівалентних схем (**RC-моделі**) надають можливість моделювати короткочасні динаміки (падіння/відновлення напруги при переході навантаження). При ідентифікації параметрів RC-моделі можна робити більш коректні перетворення напруги у SoC з урахуванням транзієнтів.

Методи на основі машинного навчання будуються як моделі SoC/SoH від вхідних ознак (U, I, T, профіль використання, історія). Вони добре виявляють нелінійні залежності й адаптуються до специфічних профілів використання, але вимагають значних навчальних наборів даних, потребують верифікації на різних пристроях і несуть ризик overfitting.

З огляду на обмеження ноутбуків (відсутність струму у загальному доступі, різноманітність платформ), доцільно обрати наступну стратегію:

— **Базова оцінка SoC** лінійна або кусочно-лінійна апроксимація напруги з калібруванням меж $U_{\min}, U_{\max}$;

— **Компенсація навантаження**, а саме емпірична поправка ΔU, що враховує CPU%, яскравість, включені периферійні модулі (Wi-Fi, BT, dGPU) — формула і приклади далі;

— **Згладжування**, а саме легкий фільтр (наприклад, EWMA) для уникнення флапу;

— **Калібрування**, яким є процедура для визначення $U_{\min}$, $U_{\max}$при першому запуску або після заміни батареї (можливість «калібрування користувачем» через повний цикл).

— **Періодична перевірка**, якщо платформа надає струм, то за можливості включити Coulomb counting як додатковий режим.

Спрощена емпірична модель компенсації описана формулою 1.6:

$$\Delta U = k_{CPU} \cdot CPU\% + k_{br} \cdot Brightness\% + \sum_j k_j \cdot \mathbb{1}_{(module_j\ active)} \quad (1.6)$$

де ΔU – поправка напруги;

k_{CPU} – коефіцієнт для процесора;

k_{br} – коефіцієнт для яскравості;

k_j – коефіцієнт для модуля j;

$\mathbb{1}$ – індикатор увімкнення модуля (Wi-Fi, BT, dGPU тощо).

Тоді йде розрахунок ефективної напруги, яка описана формулою 1.7:

$$U_{\text{еф}} = U_{\text{поточна}} + \Delta U, \quad (1.7)$$

Де $U_{\text{еф}}$— ефективна (скоригована) напруга;

$U_{\text{поточна}}$ – поточна напруга батареї;

ΔU –поправка на навантаження.

І Оцінка SoC, яка описана формулою 1.8:

$$SoC = \text{clamp}(\frac{U_{\text{еф}} - U_{\min}}{U_{\max} - U_{\min}} \times 100\%,\ 0{,}100), \quad (1.8)$$

де SoC – рівень заряду (%);

$U_{\text{еф}}$ – ефективна напруга;

$U_{\text{мін}}$ – напруга при повному розряді;

$U_{\text{макс}}$ – напруга при повному заряді;

$\text{clamp}(x, 0{,}100)$— обмеження значення в межах 0–100%.

Орієнтовні початкові коефіцієнти можуть бути такими:

— $k_{CPU} = 0.005$ В/%(cap ≈ 0.5 В),

— $k_{br} = 0.002$ В/%(cap ≈ 0.2 В),

— Wi-Fi: $k_{wifi} = 0.01$ В(якщо активний),

— dGPU: $k_{dGPU} = 0.05$ В(приблизно; залежить від платформи).

Ці значення потрібно калібрувати під конкретну платформу.

Калібрування меж $U_{\min}, U_{\max}$ рекомендується робити або автоматично (через контрольований цикл заряд–розряд), або вручну з використанням заводських даних. Для верифікації підходу бажано порівняти результати з еталонним обладнанням (power analyzer) або з OEM-утилітами на кількох моделях ноутбуків; як метрики використовуються MAE для SoC, відносна похибка часу автономії та зміни потужності у ватах.

Переваги обраного спрощеного комбінованого підходу – це його універсальність (не вимагає струмових сенсорів), відносна простота реалізації та добрий баланс між точністю і витратами обчислювальних ресурсів, що важливо для реалізації в користувацькому програмному продукті. Обмеженням підходу є потреба калібрування емпіричних коефіцієнтів, обмежена точність на деяких ділянках кривої U–SoC та необхідність застосування додаткових моделей (RC, EKF) для коректної обробки складних транзієнтів.

У першому розділі надано докладний огляд проблематики оцінки зарядного стану акумуляторів та методів енергозбереження у контексті ноутбуків під Windows. Розглянуто фізичні причини складностей (U–SoC, внутрішній опір, температурна залежність, старіння), проаналізовано основні методи (OCV, Coulomb counting, комбіновані підходи, фільтри) та обґрунтовано вибір практичного спрощеного комбінованого підходу для реалізації EcoControl: оцінка по напрузі з емпіричними корекціями навантаження та калібруванням меж, доповнена простими заходами згладжування й опціями для подальшого інтегрування Coulomb counting за наявності даних струму.

2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ІНСТРУМЕНТІВ

У цьому розділі проведено огляд і критичний аналіз існуючих програмних рішень для моніторингу та управління енергоспоживанням у ноутбуках. Розглянуто утиліти типу BatteryCare, а також OEM-інструменти, що постачаються виробниками ноутбуків, які забезпечують базовий контроль заряду та налаштування режимів живлення. Оцінено їх функціональні можливості, інтерфейсні рішення та обмеження, пов'язані з доступом до апаратних сенсорів. Окрему увагу приділено апаратним інтерфейсам, таким як Battery Management System (BMS) та ACPI, які забезпечують низькорівневий доступ до даних про батарею, а також програмним інтерфейсам Windows – WMI та WinAPI, що дозволяють отримувати інформацію про стан живлення та застосовувати дії. Мета цього розділу полягає у виявленні сильних і слабких сторін доступних рішень, оцінці їх придатності для інтеграції в EcoControl та визначенні напрямів удосконалення, які можуть бути реалізовані у власному програмному модулі.

2.1 Огляд програмних утиліт та інструментів (BatteryCare, OEM утиліти тощо)

Програмні продукти, що працюють у сфері моніторингу акумуляторів та енергоменеджменту, можна умовно розділити на кілька груп:

- **Системні або вбудовані механізми ОС** – Power Plans, Battery Saver у Windows.
- **OEM-утиліти** – інструменти від виробника ноутбука (Lenovo Vantage, Dell Power Manager, HP Support Assistant тощо).
- **Треті сторони — споживчі утиліти** – BatteryCare, BatteryBar, HWMonitor, 3rd-party battery managers.

— **Профільні інструменти для діагностики** – AIDA64, HWiNFO, утиліти живлення для тестування (PowerMeter, зовнішні аналізатори).
— **Розробницькі інструменти / бібліотеки** – PowerCfg, WMI, WinAPI, .NET-класи для отримання даних.

BatteryCare та подібні утиліти є прикладом інструменту для користувача, який надає інтерфейс для перегляду відсотка заряду, часу до розряду/до зарядки, статистики циклів і повідомлень про стан батареї. Типові можливості таких утиліт:

— відображення поточного відсотка SoC та напруги (якщо доступно);
— відстеження циклів заряд-розряд;
— повідомлення про надмірний нагрів;
— поради щодо калібрування батареї;
— прості графіки зміни заряду у часі.

Переваги утиліт:

— інтуїтивний інтерфейс для користувача;
— легка інсталяція та мінімальні вимоги;
— корисні для нефахового моніторингу.

Обмеження:

— обмежена функціональність щодо автоматичного управління компонентами (не вимикають dGPU, не керують службами в глибину);
— залежать від даних BMS – якщо контролер дає хибні показники, покази програми також будуть некоректні;
— часто не мають можливості додаткової корекції (наприклад, корекції напруги під навантаженням).

OEM-утиліти (наприклад, Lenovo Vantage, Dell Power Manager, HP Power Manager) зазвичай мають глибшу інтеграцію з апаратним забезпеченням та розширені можливості, які включають:

— доступ до специфічних апаратних сенсорів і розширених даних BMS (якщо виробник дозволяє);

— режим «Conservation Mode» або «Battery Health Mode» (обмеження максимального заряду для продовження життя батареї);
— профілі продуктивності залежно від режимів (AC / battery);
— інструменти діагностики та оновлення firmware;
— інтеграція з драйверами GPU для перемикання dGPU / iGPU або зменшення продуктивності GPU.

Переваги:

— глибша інтеграція і, як правило, краща точність на рідній платформі;
— підтримка специфічних апаратних фіч (наприклад, керування зарядним режимом).

Недоліки:

— часто закриті, прив'язані до конкретних моделей/серій;
— великі обсяги інсталяції; можуть включати «телеметрію»;
— не універсальні; унеможливлюють легко переносити рішення для інших брендів.

Профільні діагностичні інструменти **HWiNFO, AIDA64** та подібні інструменти дають детальну інформацію про апарат (температури, напруги, струми — за наявності датчиків), зазвичай більше підходять для діагностики, ніж для адаптивного енергоменеджменту. Дані інструменти корисні для валідації експериментів та збору «правильних» даних, а також можуть показати, які дані насправді є доступними на конкретному ноутбуку, тобто які класи ACPI/BMS реалізовані.

У підсумку ці утиліти корисні для інформування, OEM-утиліти надають найкращі можливості на своїй платформі, а діагностичні інструменти потрібні для верифікації та збору еталонних даних. Для EcoControl доцільно поєднати універсальність (базовий режим, який працює на всіх моделях) з можливістю підключати розширені модулі/плагіни для платформ із додатковим API.

2.2 Обмеження апаратних інтерфейсів та джерел даних (BMS, ACPI)

Загальним оглядом апаратних джерел даних для визначення стану акумулятора та пов'язаних параметрів є:

- **Battery Management System (BMS)** – апаратний контролер батареї, що встановлює рівень заряду, надає оцінки SoC/SoH, час роботи, статус зарядки. BMS інколи містить власну логіку калібрування.
- **ACPI (Advanced Configuration and Power Interface)** – стандартна прошаркова система для обміну інформацією про живлення між апаратним забезпеченням і ОС. Відкриває набір методів та властивостей, які ОС читає.
- **SMBus / I2C датчики** – у деяких схемах BMS доступна телеметрія через SMBus; це вимагає відповідних драйверів.
- **Драйвери виробника** – специфічні драйвери або служби, що надають розширену інформацію.

Типові поля, які може надавати BMS/ACPI:

- Рівень заряду (BatteryRemainingPercent, EstimatedChargeRemaining)
- Статус (Charging/Discharging/Full)
- Поточна напруга батареї (Voltage)
- Поточний струм (Current) – доступний не завжди
- Оцінка часу до розрядки/до зарядки (EstimatedRunTime, EstimatedChargeTime)
- Температура батареї
- Цикли заряд-розряд, існуюча повна ємність (DesignCapacity, FullChargeCapacity)
- Відомості про виробника, серійний номер

Однак поля можуть бути відсутні або заповнюватись некоректно (особливо після ремонту) та часто струм або точні дані є недоступними через політику виробника або апаратні обмеження.

Проблеми та обмеження BMS і ACPI, включають:

— Неповнота даних, а саме необхідні поля (наприклад, фактичний струм) часто не доступні через стандартні інтерфейси. Це ускладнює застосування Coulomb counting або складних комбінованих методів.
— Багато виробників материнських плат/акумуляторів реалізують ACPI по-різному: різні імена полів, різні одиниці виміру, різна частота оновлення.
— Після ремонту батареї або заміни банок BMS може бути некоректно налаштований, що призводить до неправильних відсоткових показів у Windows (інколи 0%).
— Деякі розширені дані доступні лише через OEM-драйвери/служби, яких немає на інших моделях.
— Затримки оновлення, тобто коли дані від BMS можуть оновлюватись рідше, ніж потрібно для адаптивного контролю в реальному часі (latency).

На таблиці 2.1 зображено приклад наявності параметрів на різних на ноутбуках.

Таблиця 2.1

Доступність параметрів на різних ноутбуках

Параметр	**Зазвичай доступний через ACPI/WMI**	**Наявність (типово)**
Відсоток заряду (SoC)	Так	Так
Напруга батареї (Voltage)	Часто	Часто
Струм (Current)	Рідко	Ні
Температура батареї	Часто	Опційно
Повна ємність (FullChargeCapacity)	Часто	Опційно
Час до розряду/зарядки (EstimatedRunTime)	Часто	Так, але неточно
Індивідуальні банки / баланс	Ні	Ні (переважно тільки у сервісних інструментах

Таким чином, можна зробити висновок, що для EcoControl слід передбачити універсальний режим, що працює на основі напруги та загальнодоступних параметрів (CPU%, яскравість, стан мережі), а також розширений режим для платформ з додатковими API (де доступні температури, SoH). Також необхідні процедури калібрування меж $U_{\min}, U_{\max}$ і механізми фільтрації для компенсації затримок і шуму у даних.

2.3 Можливості та обмеження WinAPI / WMI для збору даних про живлення

У Windows для збору даних про живлення доступні такі інтерфейси:

- **WMI (Windows Management Instrumentation)** – це класичний механізм для запиту системної інформації (Win32_Battery, Win32_PortableBattery, BatteryStaticData тощо).
- **Win32 API / Power Management API** – низькорівневі функції, які дозволяють підписуватися на події і читати статус (GetSystemPowerStatus, RegisterPowerSettingNotification).
- **.NET wrappers** – System.Management для WMI, Microsoft.Win32.SystemEvents для подій PowerModeChanged.
- **Powercfg.exe** – CLI-утиліта для роботи з планами живлення.
- **OEM SDK / драйвери** розширені можливості, недокументовані API.

Запити через WMI (наприклад, **Win32_Battery**) дозволяють отримати поля EstimatedChargeRemaining, BatteryStatus, EstimatedRunTime, BatteryRechargeTime, DesignCapacity, FullChargeCapacity.

Приклад PowerShell запиту:

```
Get-WmiObject -Class Win32_Battery | Format-List *
```

Приклад C# (ManagementObjectSearcher):

```
var searcher = new ManagementObjectSearcher("SELECT * FROM Win32_Battery");
foreach (ManagementObject obj in searcher.Get())
```

```
{
    Console.WriteLine("EstimatedChargeRemaining:" +
obj["EstimatedChargeRemaining"]);
    Console.WriteLine("Voltage: " + obj["Voltage"]);
    Console.WriteLine("BatteryStatus: " + obj["BatteryStatus"]);
}
```

Обмеженнями WMI є повільні запити швидкості real-time, поля можуть мати null або нереалістичні значення залежно від BMS та WMI не гарантує наявності поля Current (струм).

GetSystemPowerStatus надає базову інформацію статусу живлення, а саме:

- ACLineStatus – живлення від мережі або акумулятор;
- BatteryFlag – індикатори;
- BatteryLifePercent – відсоток заряду;
- BatteryLifeTime – час до розряду.

Приклад (C# P/Invoke):

```
[DllImport("kernel32")]
static extern bool GetSystemPowerStatus(out SYSTEM_POWER_STATUS
sps);
SYSTEM_POWER_STATUS status;
if (GetSystemPowerStatus(out status))
{
    Console.WriteLine(status.BatteryLifePercent);
}
```

Переваги:

- Легко викликати, багатовідомчий.

Недоліки:

- Дуже базова інформація;
- Немає напруги, струму, температури в детальному вигляді.

WM_POWERBROADCAST надає повідомлення, яке отримують віконні програми при зміні режиму живлення; У WM_POWERBROADCAST одним із параметрів є PBT_APMPOWERSTATUSCHANGE. Це дає можливість реагувати на зміну живлення без постійного опитування.

Приклад у WinForms (C#):

```
protected override void WndProc(ref Message m)
{
   const int WM_POWERBROADCAST = 0x218;
   const int PBT_APMPOWERSTATUSCHANGE = 0xA;
   if (m.Msg == WM_POWERBROADCAST && (int)m.WParam == PBT_APMPOWERSTATUSCHANGE)
   {
      // Перевірити статус через GetSystemPowerStatus або WMI
   }
   base.WndProc(ref m);
}
```

Microsoft.Win32.SystemEvents.PowerModeChanged – це подія, яку можна підписати на початку роботи додатку:

```
Microsoft.Win32.SystemEvents.PowerModeChanged += (s, e) => {
   if (e.Mode == PowerModes.StatusChange)
   {
      // реагувати
   }
};
```

Переваги подій:

— Ефективні (не потребують таймерів polling);

— Дають швидку реакцію на зміни.

Обмеження:

— Подія повідомляє про зміну статусу, але не дає детальних даних — їх треба додатково отримувати через WMI/WinAPI.

Windows API також дозволяє програмно змінювати параметри планів живлення, наприклад через PowerWriteACValueIndex, PowerSetActiveScheme. Це дає можливість EcoControl змінювати Power Plans або конкретні параметри CPU/performance.

Приклад PowerShell:

```
# встановити активний план
powercfg /setactive <GUID>
```

Зміна глобальних планів вплине на всю систему, тому краще змінювати лише конкретні налаштування або використовувати «локальні» заходи, наприклад керування TurboBoost, яскравістю, окремими службами.

Прикладами конкретних WMI/WinAPI запитів, корисних для EcoControl є:

- Win32_Battery – EstimatedChargeRemaining, Voltage, BatteryStatus.
- WMI для моніторингу дисплею – WmiMonitorBrightness (зчитування яскравості).
- WMI для мережі – MSFT_NetAdapter / Win32_NetworkAdapter (виявлення стану Wi-Fi/BT адаптерів).
- Процесор – зчитування завантаження CPU через PerformanceCounter або Win32_Processor (LoadPercentage).
- Device management – devcon або SetupAPI для управління пристроями (увімкнення/вимкнення dGPU або адаптерів).

Обмеженням API є те, що:

- Різні версії Windows мають різну поведінку API (Edge cases).
- Доступ до деяких дій потребує прав адміністратора (вимикання служб, зміна драйверних параметрів, виклик devcon тощо).
- Частина операцій (відключення dGPU) може призвести до невизначеної поведінки або вимоги перезавантаження/перепідключення драйверів.

Рекомендаціями для EcoControl є:

— Забезпечити режим роботи без прав адміністратора (ті операції, що вимагають прав, робити через віддалені служби або з вимогою підвищення прав при необхідності).
— Використовувати події (WM_POWERBROADCAST/SystemEvents) замість таймерного опитування для ефективності.
— Робити «безпечний» набір дій: починати з неглибоких оптимізацій (яскравість, EcoQoS) і лише при потребі й з дозволу користувача застосовувати більш інвазивні дії (відключення служб, dGPU).
— Логувати всі зміни з можливістю відкату (undo) у разі проблем.

Архітектурно збір даних у EcoControl можна організувати у вигляді рівнів:

— Data Layer: WMI / WinAPI / OEM SDK — збір сирих даних.
— SensorManager: нормалізація, калібрування, базова корекція (ΔU).
— SoCModel: обчислення SoC.
— ModeController: логіка порогів і дій.
— ActionExecutor: застосування налаштувань (яскравість, служби, devcon).
— UI / Logger: відображення, збереження логів.

Практичні сценарії застосування включають:

— Нотифікація на заміну батареї: якщо FullChargeCapacity значно нижче DesignCapacity → порадити користувачу заміну/калібрування.
— Режим аварійного енергозбереження: при SoC < 15% — застосувати агресивні дії (вимкнути Wi-Fi, BT, очистити RAM, обмежити CPU) — попередньо повідомивши користувача.
— Розширений режим для OEM: використати додаткові сенсори/дані (струм)
— Перейти на Coulomb counting і покращену оцінку SoC.

Таким чином, можна зробити висновок, що WMI/WinAPI дають базовий набір даних, достатній для універсальної реалізації EcoControl у «універсальному» режимі. Для підвищення точності бажано підтримати «розширені модулі» для OEM-платформ із доступом до струму/додаткової

телеметрії, а також необхідно врахувати обмеження прав та можливі наслідки інвазивних дій (вимкнення пристроїв/служб).

У **другому розділі** проведено ґрунтовний аналіз програмних інструментів, апаратних інтерфейсів та системних API, релевантних для задач енергозбереження у ноутбуках. Основними висновками є те, що:

1. **Програмні утиліти** мають корисний набір функцій, але лише OEM-утиліти часто володіють реальними апаратними перевагами на своїй платформі; утиліти третьої сторони — корисні для інформування користувача, але не для комплексного автоматичного керування компонентами.
2. **BMS та ACPI** – це основні джерела даних, проте вони часто неповні, непослідовні та залежать від виробника; особливої уваги потребують питання доступності струму та якості EstimatedRunTime.
3. **WMI/WinAPI** – достатні для універсальної реалізації EcoControl, але потребують консервативного підходу: події замість опитування, безпечні дії, логування і можливість розширення функціоналу при наявності OEM API.
4. Архітектура EcoControl має бути багаторівневою, з базовим «універсальним» режимом і додатковими «плагінами» для платформ з розширеними можливостями.

3 РОЗРОБКА МАТЕМАТИЧНОЇ МОДЕЛІ ТА МОДИФІКОВАНОГО МЕТОДУ

У цьому розділі викладено розроблену математичну модель оцінки стану заряду акумулятора (SoC) на основі напруги з урахуванням корекції на навантаження. Модель враховує вплив активності процесора та яскравості дисплея на просадку напруги, що дозволяє отримати більш точне значення заряду у порівнянні з класичним методом відкритої напруги. Описано модифікований адаптивний метод енергозбереження, який поєднує оцінку SoC з логікою прийняття рішень щодо перемикання режимів живлення. Алгоритмічна частина включає відповідність дій обраним чекбоксам користувача, що забезпечує інтеграцію між інтерфейсом та внутрішньою логікою системи. Окремо розглянуто процедуру калібрування параметрів, яка дозволяє адаптувати модель до конкретного пристрою, враховуючи його апаратні особливості. Також наведено технічні аспекти реалізації, що включають використання C#, WinForms та взаємодію з системними API для збору даних і застосування дій.

3.1 Постановка задачі та вимоги

Постановка задачі полягає у створенні математичної моделі та адаптивного алгоритму, які здатні забезпечувати коректну оцінку стану заряду акумулятора на ноутбуках навіть за умов обмеженого доступу до апаратних даних, зокрема за відсутності достовірних показників струму. Алгоритм повинен автоматично приймати та виконувати енергозберігаючі дії, наприклад обмеження або вимкнення окремих компонентів, враховуючи поточний стан заряду, рівень навантаження та поведінку користувача. Важливою **вимогою** є забезпечення універсальності роботи на різних апаратних платформах та можливість підвищення точності на системах, які мають доступ до розширених OEM-даних.

Такі **вимоги** висувають кілька ключових критеріїв до майбутньої системи. Модель має працювати з мінімальними апаратними залежностями, використовуючи принаймні дані про напругу батареї, завантаження центрального процесора та яскравість дисплея. Важливою характеристикою алгоритму є його адаптивність, що передбачає наявність порогів та гістерезису для запобігання надмірно частим перемиканням режимів, які могли б знижувати якість роботи пристрою. Безпека роботи системи полягає в тому, що будь-які дії, які потенційно можуть призвести до втрати даних, зокрема примусове завершення процесів, повинні виконуватися лише після відповідного попередження користувача. Передбачена й можливість валідації шляхом проведення калібрування та ведення логів, що дозволить аналізувати роботу системи та коригувати її параметри.

3.2 Базова модель перетворення напруги в SoC

Нехай $U_{\text{поточна}}$— напруга на клемах акумулятора (виміряна через WMI/ACPI), $U_{\min}$ і $U_{\max}$ – це напруги, які відповідають 0% та 100% відповідно (виміряні або заводські значення). Базова лінійна оцінка SoC, яка описана формулою 3.1:

$$SoC_{base} = \frac{U_{\text{поточна}} - U_{\min}}{U_{\max} - U_{\min}} \times 100\%, \quad (3.1)$$

Де $U_{\text{поточна}}$ – поточна напруга батареї;

$U_{\text{мін}}$— напруга при повному розряді;

$U_{\text{макс}}$— напруга при повному заряді.

Базова модель оцінки SoC на основі напруги є простим і ефективним способом визначення залишкового заряду акумулятора без потреби в складних сенсорах або інтеграції струму. Вона особливо корисна у випадках, коли доступні лише дані про напругу, отримані через WMI або ACPI.

Щоб уникнути виходів за межі, описано формулу 3.2:

$$SoC_{base} = \text{clamp}(SoC_{base},\ 0,\ 100), \quad (3.2)$$

Де $\text{clamp}(x, 0{,}100)$ – обмеження значення в межах 0–100%.

Цей підхід дуже простий і зрозумілий, однак не враховує падіння напруги під навантаженням, тому вводимо поправку ΔU, що коригує $U_{\text{поточна}}$до «ефективного» значення $U_{\text{еф}}$, яке наближено відповідає OCV та описано формулою 3.3:

$$U_{\text{еф}} = U_{\text{поточна}} + \Delta U, \tag{3.3}$$

Де $U_{\text{еф}}$ – ефективна (скоригована) напруга;

ΔU – поправка на навантаження.

Емпірична модель для ΔU, описана формулою 3.4:

$$\Delta U = k_{CPU} \cdot CPU\% + k_{br} \cdot Brightness\% + \sum_{m \in M} k_m \cdot s_m, \tag{3.4}$$

Де $CPU\%$ – відсоток завантаження CPU (0–100);

$Brightness\%$ – відсоток яскравості дисплея (0–100);

M – множина активних модулів (Wi-Fi, Bluetooth, dGPU та ін.);

$s_m \in \{0,1\}$ – індикатор активності модуля m;

k_{CPU}, k_{br}, k_m – емпіричні коефіцієнти (вольти/одиниця).

Приклад рекомендованих значень (рекомендація за результатами попередніх експериментів):

— $k_{CPU} = 0.005$ В/%(максимум 0.5 В),

— $k_{br} = 0.002$ В/%(максимум 0.2 В),

— $k_{wifi} = 0.01$ В,

— $k_{bt} = 0.005$ В,

— $k_{dGPU} = 0.05$ В.

Сумарне ΔUмає бути обмежене (cap), наприклад, яке описано формулою 3.5, щоб уникнути нереалістичних компенсацій:

$$\Delta U \leq \Delta U_{max} = 0.7 \text{ В}, \tag{3.5}$$

Де ΔU_{max}— максимальна допустима поправка;

0.7 В — емпірично обрана межа для уникнення перенасичення моделі.

Тому після корекції отримаємо остаточну формулу, яка описана формулою 3.6:

$$SoC = \text{clamp}(\frac{U_{\text{еф}} - U_{\min}}{U_{\max} - U_{\min}} \times 100\%, 0, 100), \quad (3.6)$$

Де $U_{\text{еф}}$ — ефективна напруга;

$U_{\text{мін}}, U_{\text{макс}}$ — межі заряду;

$\text{clamp}(x, 0{,}100)$ — обмеження значення в межах 0–100%.

Цей SoC використовується для ухвалення рішень.

Щоб уникнути швидких коливань SoC через миттєві спади або підйоми навантаження, в майбутньому було б доцільно застосувати експоненційне згладжування (EWMA), яке описано формулою 3.7:

$$SoC_t = \alpha \cdot SoC_{\text{новий}} + (1 - \alpha) \cdot SoC_{t-1}, \quad (3.7)$$

Де SoC_t – згладжене значення на поточному кроці;

$SoC_{\text{новий}}$ – нове обчислене значення;

α – коефіцієнт згладжування (0.3–0.5 рекомендовано).

Логікою прийняття рішень (ModeController) є логіка з трьома основними зонами:

— **Performance Zone:** $SoC > T_{high}$ (наприклад, $T_{high} = 50\%$) → продуктивний режим.

— **Normal Zone:** $T_{low} \leq SoC \leq T_{high}$ (наприклад, $T_{low} = 15\%$) → поміркована оптимізація.

— **Critical Zone:** $SoC < T_{low}$ → максимальні заходи енергозбереження.

Щоб уникнути частих перемикань, застосовуємо *гістерезис* (hysteresis), тобто різні входи/виходи зон. Наприклад: при вході в Critical Zone $T_{low\_enter} = 15\%$, при виході $T_{low\_exit} = 18\%$. Дії для зон представлено у вигляді таблиці 3.1.

Таблиця 3.1

Дії для зон

Зона	**Приклад дій (за замовчуванням)**
Performance	Залишити всі служби, максимальна продуктивність CPU, EcoQoS off

Продовження таблиці 3.1

Дії для зон

Зона	**Приклад дій (за замовчуванням)**
Normal	Зменшити background tasks, обмежити TurboBoost за умовою, знизити яскравість на 10–20%
Critical	Вимкнути Wi-Fi/BT (за чекбоксом), відключити dGPU, зупинити неналежні служби, почистити RAM на N%, відключити TurboBoost, попередити користувача

Усі дії мають бути керовані чекбоксами, які користувач може увімкнути або вимкнути.

У системі кожен елемент керування, зокрема чекбокс, відповідає певному набору дій, які виконує програма. Це може стосуватися як відображення інформації, так і застосування конкретних змін до роботи апаратних компонентів. Наприклад, приблизний відсотковий рівень заряду відповідає поданому користувачу значенню SoC, тоді як показники напруги батареї поєднують відображення поточних вимірювань та розрахункового ефективного значення. Вимикання режиму TurboBoost може здійснюватися через доступні утиліти або відповідні регістри процесора, тоді як зміна яскравості реалізується за допомогою доступних методів керування монітором через інтерфейси Windows.

У системі також реалізована можливість активації режиму енергоефективності, який використовує стандартні політики живлення Windows або API для управління якістю обслуговування, коли це доступно. Додатково можуть застосовуватися дії для очищення оперативної пам’яті, завершення другорядних фонових процесів або керування службами операційної системи, при цьому передбачено механізм безпечного зупинення, що уникає втручання у критичні компоненти. У разі потреби система може вимикати дискретну графічну карту через спеціалізовані інструменти або API виробника, хоча окремі

платформи можуть вимагати перезавантаження для коректного переключення. Серед інших можливостей — вимкнення бездротових інтерфейсів у періоди бездіяльності, що здійснюється через відповідні інструменти керування мережею. Усі дії супроводжуються журналюванням і можливістю відкату, що є невід'ємною частиною зручності та безпечності користувацького досвіду.

3.3 Алгоритми та їх графічне представлення

У цьому розділі представлено логіку роботи системи енергозбереження EcoControl, яка реалізує адаптивне керування режимами живлення ноутбука на основі аналізу стану акумулятора, навантаження на процесор, яскравості дисплея та активності користувача. Для візуалізації процесів було створено низку діаграм, що демонструють послідовність дій, умови переходу між режимами та загальну архітектуру алгоритму.

Алгоритм роботи системи починається з процесу ініціалізації, під час якого завантажуються всі необхідні конфігураційні параметри, зокрема коефіцієнти моделі, межі напруги батареї, порогові значення для визначення зон заряду, а також параметри гістерезису, які дозволяють уникнути частих перемикань між режимами. Після завершення ініціалізації система переходить у режим очікування подій або таймерів, які запускають чергову ітерацію аналізу.

Під час кожного циклу алгоритм зчитує дані сенсорів: поточну напругу акумулятора, навантаження процесора, яскравість дисплея та стан модулів. На основі цих параметрів виконується розрахунок скоригованої напруги, яка моделює стан батареї без навантаження. Для цього використовується формула, яка описана формулою 3.8:

$$V\text{кор} = V + CPU\% * 0.005 + Brightness\% * 0.002, \quad (3.8)$$

Де V — поточна напруга;

CPU% - навантаження процесора у відсотках;

Brightness% — яскравість дисплея.

Скоригована напруга дозволяє компенсувати тимчасові просадки, спричинені активністю системи, і забезпечити більш точну оцінку залишкового заряду. Якщо отримане значення перевищує фізичний максимум для літій-іонної батареї (12.6 В), воно обмежується до цього порогу.

На основі скоригованої напруги розраховується відсотковий рівень заряду акумулятора за формулою нормалізації. Далі застосовується метод згладжування (наприклад, EWMA), який стабілізує результат і зменшує вплив короткочасних коливань. Після цього система визначає поточну зону роботи: продуктивну (SoC > 50%), нормальну (SoC 20–50%) або критичну (SoC < 20%).

У відповідності до визначеної зони та активованих користувачем параметрів, система формує перелік дій, які слід виконати. У продуктивному режимі активується максимальна продуктивність процесора, вимикаються обмеження, а система працює у звичайному режимі. У нормальному режимі активується енергозбереження: знижується продуктивність CPU, активується паркування ядер, вимикаються фонові процеси, застосовується EcoQoS, зменшується яскравість дисплея. У критичному режимі додатково вимикаються візуальні ефекти, GPU, Wi-Fi та Bluetooth при бездіяльності, очищується RAM та деактивуються непотрібні служби.

Такий підхід дозволяє забезпечити адаптивну реакцію системи на зміну умов навантаження, підтримуючи баланс між продуктивністю та енергоефективністю. Завдяки цьому користувач отримує стабільну роботу пристрою навіть у динамічних сценаріях використання. Крім того, алгоритм враховує індивідуальні налаштування, що робить його гнучким та придатним для різних профілів роботи. У результаті EcoControl виступає як інтелектуальний посередник між апаратними можливостями та потребами користувача, забезпечуючи оптимальний режим функціонування системи.

Для демонстрації логіки роботи системи було створено кілька діаграм, кожна з яких детально описує окремий аспект алгоритму енергозбереження. На рисунку 3.1 зображена діаграма послідовності енергозбереження Windows.

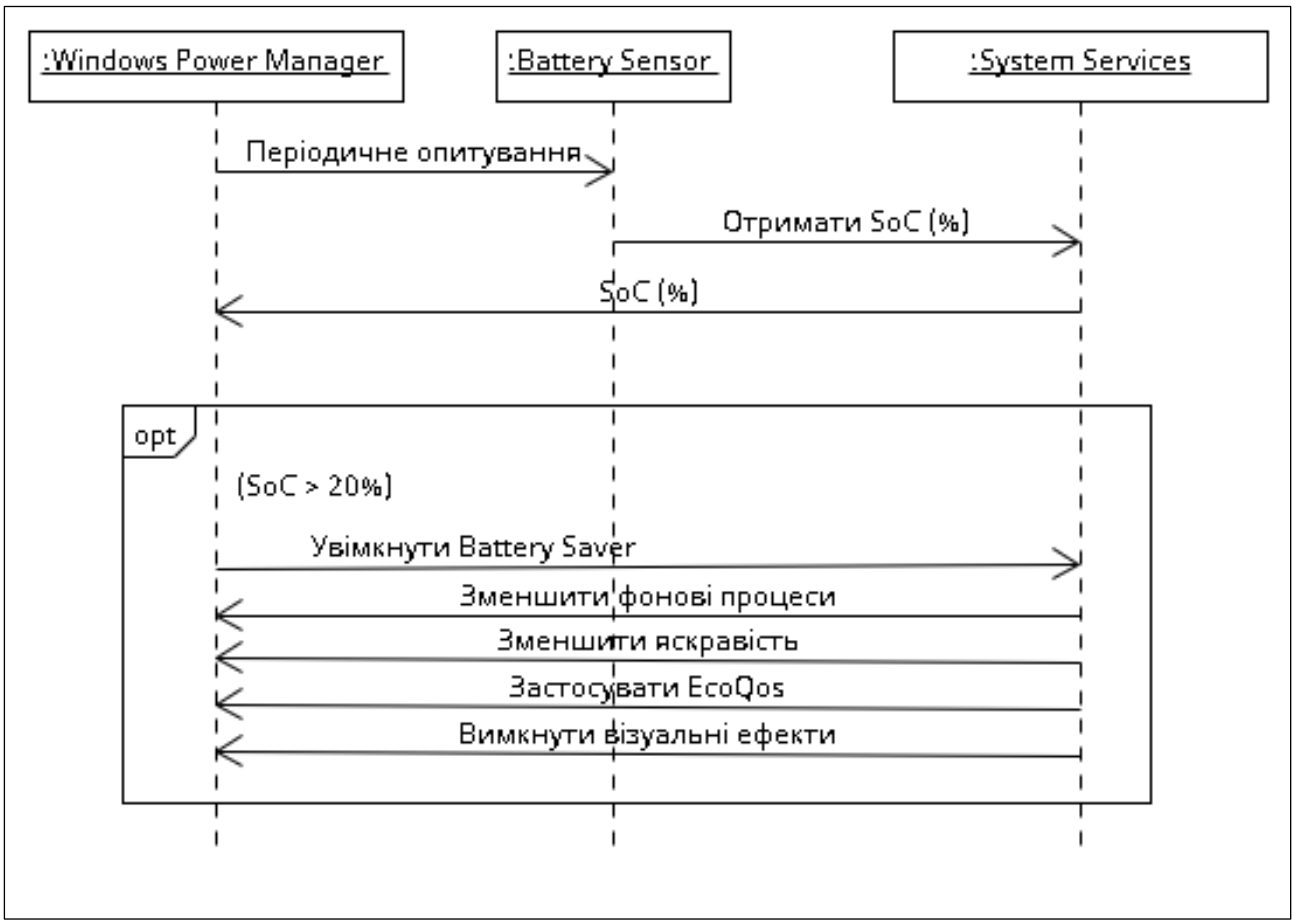


Рис. 3.1 Діаграма послідовності енергозбереження Windows

Ця діаграма ілюструє стандартну логіку переходу в режим Battery Saver при низькому рівні заряду акумулятора. Вона показує, як система реагує на досягнення критичного порогу заряду, активуючи енергозберігаючі механізми, такі як зниження яскравості дисплея, обмеження продуктивності процесора та відключення непотрібних служб. Діаграма демонструє послідовність подій від виявлення низького заряду до активації режиму, що дозволяє зрозуміти базову поведінку операційної системи у стандартних умовах.

На рисунку 3.2 зображена діаграма послідовності інтелектуального енергозбереження в EcoControl.

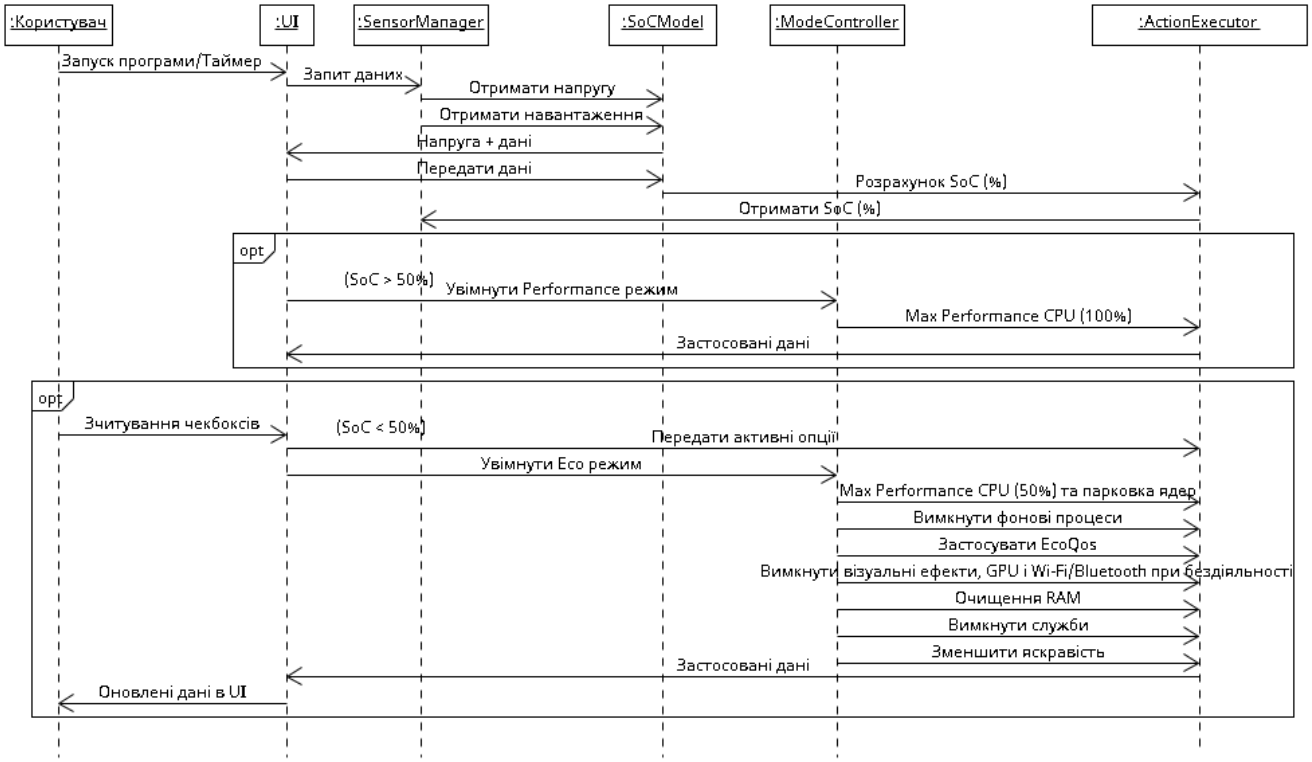


Рис.3.2 Діаграма послідовності інтелектуального енергозбереження Windows

Ця діаграма розкриває розширену логіку системи EcoControl, яка враховує додаткові параметри, такі як навантаження на процесор, яскравість дисплея та активність користувача. Вона демонструє, як система адаптує свої дії відповідно до реального навантаження, забезпечуючи більш гнучке та ефективне керування енергоспоживанням. Діаграма включає умови переходу між режимами, а також дії, що виконуються у кожному режимі, що допомагає краще зрозуміти інтелектуальний підхід EcoControl.

На рисунку 3.3 зображена схема інтелектуального енергозбереження в EcoControl.

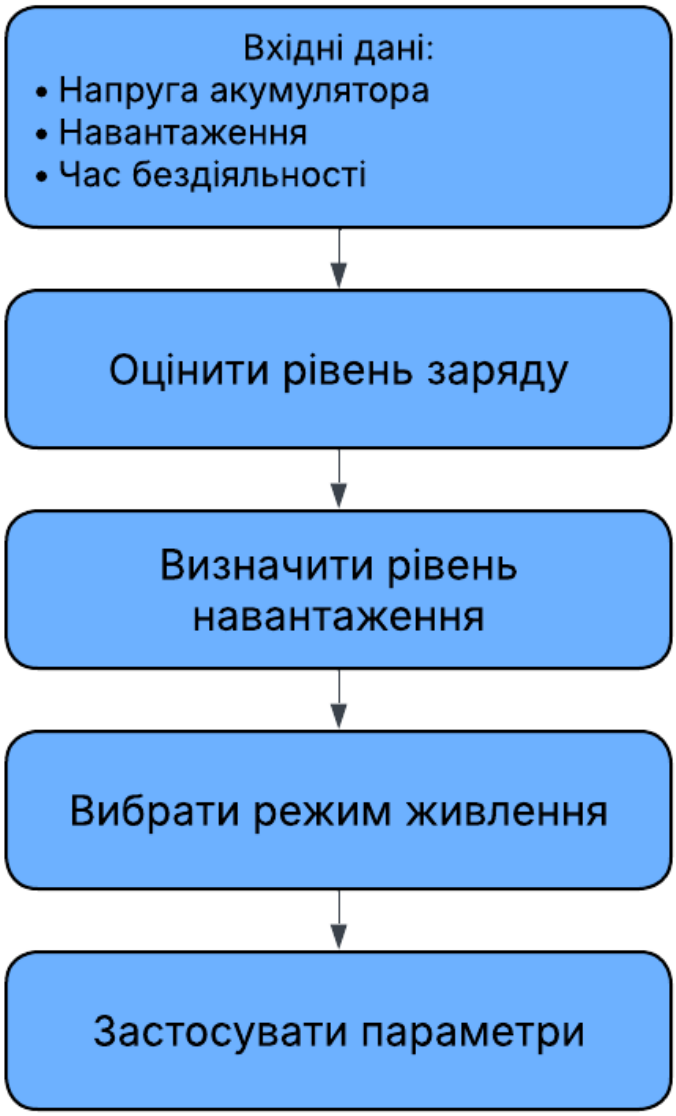


Рис.3.3 Схема інтелектуального енергозбереження Ecocontrol

Ця схема узагальнює архітектуру алгоритму, починаючи з вхідних даних таких як напруга акумулятора, навантаження процесора, яскравість дисплея і завершується застосуванням відповідних параметрів для керування режимами живлення. Вона показує основні компоненти системи, їх взаємодію та послідовність обробки інформації, що дозволяє оцінити загальний дизайн та структуру алгоритму.

На рисунку 3.4 зображений алгоритм адаптивного керування енергоспоживанням із урахуванням параметрів, встановлених користувачем.

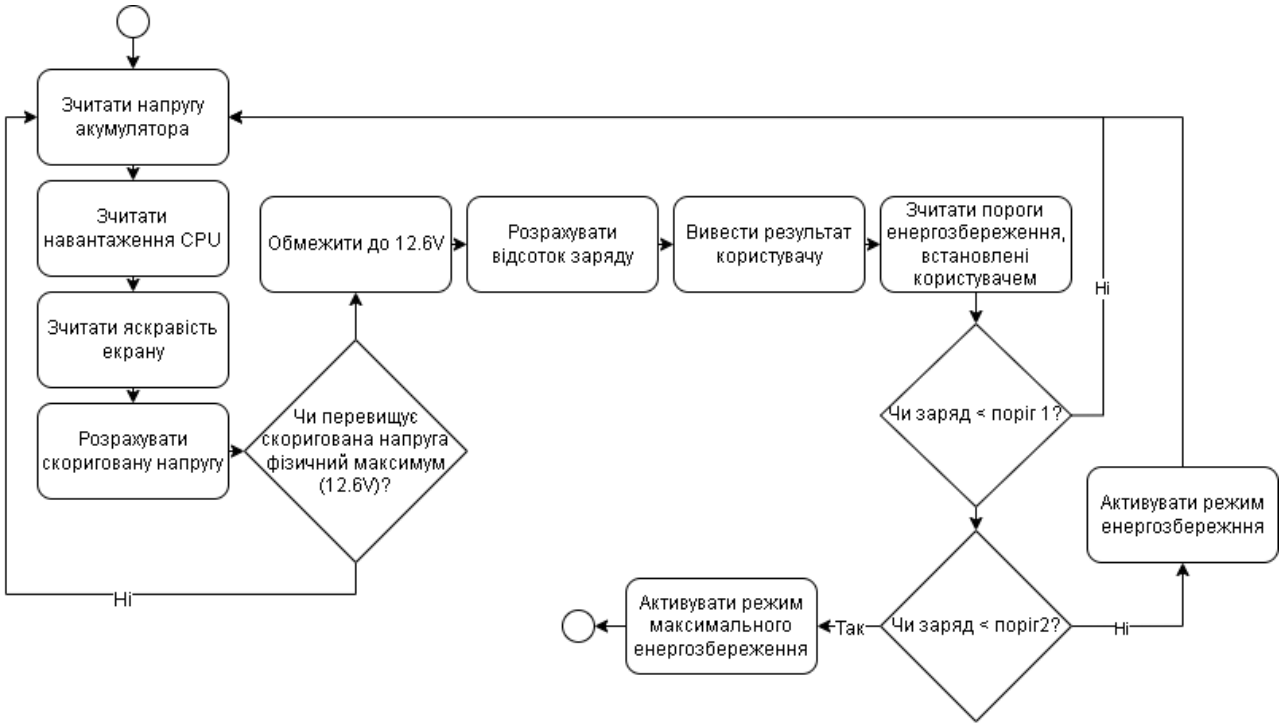


Рис.3.4 Алгоритм адаптивного енергозбереження на основі налаштувань користувача

Цей алгоритм реалізує адаптивне керування режимами енергоспоживання залежно від рівня заряду акумулятора та параметрів, встановлених користувачем. На основі зчитаних апаратних даних (напруга, навантаження CPU, яскравість екрану) та заданих порогів система приймає рішення про активацію режиму енергозбереження або максимального енергозбереження. Після виконання відповідних дій інтерфейс оновлюється, зміни логуються, і система повертається в режим очікування. Такий підхід дозволяє забезпечити баланс між продуктивністю та автономністю пристрою, враховуючи як технічні параметри, так і поведінкові налаштування користувача. Візуалізація алгоритму у вигляді блок-схеми дозволяє чітко відстежити логіку прийняття рішень, що є важливим для реалізації, тестування та подальшого вдосконалення програмного модуля.

3.4 Реалізація, практичні поради (C# / WinForms)

Архітектура модулів:

- **SensorManager** — абстракція для збору: WMI, PerformanceCounter, P/Invoke, OEM SDK. Має кеш і обробку помилок.
- **SoCModel** — клас з методами ComputeSoC(U, cpu, br, modules).
- **ModeController** — логіка порогів і гітерезису.
- **ActionExecutor** — набір команд із правами: brightness API, ServiceController, Process.Kill (safely), devcon invocation, PowerWriteACValueIndex.
- **UI** — головне вікно, віджет, лог виводу.
- **Logger** — збереження CSV/JSON логів для подальшого аналізу.

Приклади викликів:

- Зчитування напруги: WMI Win32_Battery.Voltage або GetSystemPowerStatus (не дає Voltage).
- Яскравість: WmiMonitorBrightness / WmiMonitorBrightnessMethods.
- CPU%: PerformanceCounter("Processor", "% Processor Time", "_Total").

Для дій, що потребують прав адміністратора, реалізувати окремий сервіс з зупинкою служб, виклик devcon. Рекомендується просити elevate при першому запуску або виконувати інвазивні дії через окремий сервіс, який запускається з elevated rights, а також обов'язково робити підтвердження перед виконанням дій, що можуть порушити роботу користувача.

3.5 Налаштування параметрів та процедура калібрування

Правильне **налаштування параметрів** та проведення **калібрування** є важливою частиною роботи моделі. Визначення меж напруги, зокрема максимального та мінімального значень, може відбуватися автоматично під час першого запуску системи. Максимальна напруга зазвичай фіксується у момент повного заряду акумулятора або на основі даних про проектну напругу, які надає

контролер батареї. Мінімальне значення визначається або за заводськими параметрами, або під час контрольованого розряду до безпечного рівня. Користувачеві може бути рекомендовано один цикл повного заряду та розряду для підвищення точності встановлених параметрів.

Підбір коефіцієнтів, які впливають на оцінку стану заряду, здійснюється на основі емпіричних спостережень та тестувань у різних режимах роботи, таких як простій, офісні завдання або пікове навантаження. Для кожного сценарію фіксуються значення напруги, споживання енергії та час автономної роботи. Порівняння цих даних з еталонними значеннями дозволяє підібрати коефіцієнти таким чином, щоб мінімізувати відхилення оцінки SoC. Окрему роль у стабільності системи відіграють пороги та гістерезис, які визначають межі переходу між режимами та запобігають помилковим спрацьовуванням. Важливо також враховувати час стабілізації, через що для інвазивних дій рекомендується використовувати затримку перед активацією.

Під час **тестування** система оцінюється за кількома основними метриками, серед яких середня похибка оцінювання стану заряду, зміна енергоспоживання у порівнянні зі звичайним режимом роботи та приріст автономності в реальних умовах. Також важливим критерієм є мінімізація зайвих перемикань режимів. Успішною вважається така конфігурація, що забезпечує похибку не більше десяти відсотків, зменшує енергоспоживання та дозволяє отримати приріст часу автономної роботи не менше ніж на півтори години.

У контексті оцінки **надійності** запропонованої моделі важливо враховувати, що робота за умов обмежених даних неминуче накладає певні обмеження на точність прогнозування. Це означає, що система потребує періодичного калібрування для підтримання стабільної та достовірної роботи. Також суттєвим елементом є забезпечення логування дій та журналів, що дозволить проводити подальший аналіз і вчасно виявляти можливі відхилення або проблеми у функціонуванні.

Разом з тим існують певні **обмеження підходу**. Насамперед це стосується того, що емпірично підібрані коефіцієнти не завжди будуть коректно працювати

на всіх апаратних платформах, оскільки різні системи характеризуються унікальними параметрами електроживлення та поведінкою батарей.

Окремі операції, наприклад вимкнення дискретної графічної карти, можуть бути потенційно ризиковими або навіть недоступними на деяких пристроях. Ще одним суттєвим фактором є те, що відсутність інформації про струм споживання обмежує можливість використання методів, заснованих на Coulomb counting, що традиційно забезпечують вищу точність оцінки стану заряду.

Подальший розвиток системи може включати кілька напрямів **удосконалення**. Одним із них є застосування адаптивних фільтрів, наприклад EWMA або інших механізмів згладжування, що дозволить підвищити стабільність оцінки стану заряду. За наявності доступу до даних про струм споживання система зможе інтегрувати методологію Coulomb counting, що значно поліпшить точність.

Перспективним також є використання розширеного чи класичного фільтра Кальмана для об’єднання даних про напругу, струм та температуру у випадках, коли такі параметри доступні. Машинне навчання може бути застосоване для автоматичного визначення індивідуальних коефіцієнтів моделі під конкретну апаратну платформу, що знизить залежність від ручного налаштування. Окрім цього, система може бути доповнена механізмами автоматичної діагностики та попередження про некоректну роботу підсистеми керування батареєю, що підвищить її надійність і робитиме роботу користувача безпечнішою та прогнозованішою.

У третьому розділі представлено розроблену математичну модель оцінки SoC на основі напруги з урахуванням емпіричної компенсації навантаження та обґрунтовано вибір спрощеного комбінованого підходу як компромісу між практичністю і точністю. Описано модифікований метод адаптивного енергозбереження з чіткою логікою прийняття рішень, набором дій, алгоритмом (псевдокод), рекомендаціями з калібрування та методикою тестування. Наведені практичні поради щодо реалізації у C# / WinForms, питання безпеки та обмеження підходу. Рекомендовано подальші шляхи удосконалення (EWMA, EKF, ML-тюнінг).

4 РЕАЛІЗАЦІЯ ТА ОЦІНКА ПРОГРАМНОГО МОДУЛЯ EcoControl (архітектура, реалізація, тестування, результати, обговорення)

У цьому об’єднаному розділі наведено опис реалізації програмного модуля EcoControl, який поєднує теоретичні напрацювання з практичною реалізацією. Розглянуто архітектуру програмного забезпечення, що складається з ключових модулів: SensorManager для збору даних, SoCModel для розрахунку заряду, ModeController для прийняття рішень та ActionExecutor для виконання дій. Детально описано реалізацію інтерфейсу користувача, який забезпечує прозору інтеграцію між діями користувача та внутрішньою логікою системи. Окрему увагу приділено механізмам збору даних через WMI та WinAPI, а також застосуванню дій через PowerShell та утиліту devcon. У розділі також викладено методику тестування, представлені та проаналізовані результати експериментів, що підтверджують ефективність запропонованого підходу. Проведено порівняльний аналіз із існуючими рішеннями, визначено переваги та обмеження EcoControl, а також сформульовано рекомендації щодо впровадження та подальшого розвитку системи. Таким чином, розділ демонструє завершений цикл — від архітектури та реалізації до тестування та оцінки результатів.

4.1 Архітектура програмного забезпечення

EcoControl реалізовано як модульну багаторівневу систему, яка складається з логічно відокремлених компонентів. Така архітектура забезпечує зручність тестування, розширення функціональності та можливість підміни окремих підсистем без перебудови всього проєкту.

Компоненти системи:

— **Data Layer (SensorManager).** Відповідає за збір сирих даних: напруга батареї, статус живлення, завантаження CPU, яскравість екрану, стани мережевих адаптерів тощо. Використовує WMI, P/Invoke (WinAPI),

PerformanceCounter та опціонально OEM SDK. Реалізує кешування та нормалізацію одиниць вимірювання.

— **SoCModel.** Містить математичну модель оцінки стану заряду (SoC). Використовує лінійну апроксимацію з компенсацією ΔU (корекція напруги залежно від навантаження), має механізми калібрування та опціонального згладжування (EWMA).

— **ModeController.** Логіка прийняття рішень: визначає поточну зону роботи (Performance / Normal / Critical) за порогами, застосовує гістерезис і debounce, формує набір дій для виконання.

— **ActionExecutor.** Виконує набір дій: зміна яскравості, застосування EcoQoS/Power Plans, відключення Wi-Fi/BT, керування службами, виклик devcon або PowerShell, очищення RAM. Забезпечує транзакційну обробку дій (виконати → перевірити → залогувати → у разі помилки відкотити).

— **UI (WinForms).** Головне вікно та трей-віджет для швидкого моніторингу: відсоток SoC, поточна напруга, скоригована напруга, поточний режим, останні логи і налаштування дій (чекбокси).

— **Logger / Storage.** Збереження логів у форматах CSV/JSON для подальшого наукового аналізу та побудови графіків.

— **Updater / PluginManager (опціонально).** Забезпечує підключення OEM-плагінів або оновлення модулів.

4.2 Опис ключових класів і модулів

Нижче наведено короткий опис основних класів (приклад для реалізації на C#) з переліком основних методів та відповідальностей.

— **SensorManager** — методи ReadVoltage(), ReadCpuUsage(), ReadBrightness(), ReadModuleStates(). Відповідає за об’єднання джерел даних, кешування і нормалізацію.

— **SoCModel** — методи ComputeDeltaU(cpu, brightness, modules), ComputeSoC(U, deltaU), Calibrate(Umin, Umax). Відповідальний за математичну частину обчислення SoC і згладжування.
— **ModeController** — методи DetermineMode(soc), GetActionsForMode(mode). Визначає пороги, застосовує гістерезис і debounce.
— **ActionExecutor** — методи ApplyActions(actionList), Rollback(actionList). Виконує дії в транзакційному режимі, логування результатів і можливість відкату.
— **UIForm** — методи UpdateDisplay(U, Ueff, soc, mode), ShowNotification(message). Забезпечує взаємодію користувача і відображення стану.
— **Logger** — методи LogEvent(type, details), ExportLogs(path). Збирає докази експериментів і дозволяє експортувати їх для подальшого аналізу.

4.3 Безпека і права

Розробка програмного модуля EcoControl передбачає виконання низки інвазивних операцій, які безпосередньо впливають на системні компоненти операційної системи Windows. До таких операцій належать: вимкнення пристроїв через утиліту *devcon*, зупинка системних служб, модифікація регістрів MSR (Model-Specific Registers), а також застосування параметрів енергозбереження, що змінюють поведінку ядра системи. З огляду на потенційні ризики, пов’язані з такими діями, питання безпеки та прав доступу є критично важливими для забезпечення стабільної та контрольованої роботи програмного забезпечення.

Інвазивні дії вимагають підвищених прав доступу, зокрема прав адміністратора або системного рівня (SYSTEM). Рекомендований підхід полягає у винесенні критичних операцій в окремий Windows Service, який працює з правами SYSTEM та взаємодіє з основним застосунком через захищений

інтерфейс. Така архітектура дозволяє ізолювати ризиковані дії від користувацького рівня, зменшуючи ймовірність випадкових або шкідливих змін. Альтернативним варіантом є використання механізму *elevate* через UAC (User Account Control), який запитує підтвердження користувача перед виконанням дій, що потребують підвищених прав.

Усі критичні дії супроводжуються логуванням, що дозволяє відстежити історію змін, виявити джерело помилки та забезпечити аудит безпеки. Для кожної операції реалізовано механізм відкату, який дозволяє повернути систему до попереднього стану у разі виникнення помилки, нестабільності або конфлікту. Наприклад, при вимкненні служби Superfetch або Windows Search зберігається її початковий стан, що дозволяє повторно активувати її при необхідності. Аналогічно, при зміні параметрів продуктивності процесора або яскравості дисплея система зберігає попередні значення, які можуть бути відновлені у разі скасування дії.

Особливу увагу приділено захисту від несанкціонованого доступу. Всі запити до системних компонентів проходять перевірку на автентичність, а доступ до критичних функцій обмежено ролями користувача. У майбутніх версіях планується інтеграція з Windows Credential Manager для зберігання токенів доступу та реалізація цифрового підпису дій, що дозволить забезпечити ще вищий рівень безпеки.

Таким чином, модуль EcoControl реалізує принципи безпечного програмного забезпечення, поєднуючи функціональність з контролем доступу, захистом від помилок та можливістю відновлення. Це дозволяє користувачу впевнено керувати енергоспоживанням системи, не ризикуючи її стабільністю або безпекою.

4.4 Інтерфейс користувача та віджети

Принципи дизайну UI є мінімалізм, інформативність і контроль. Головне вікно відображає відсоток SoC, поточну та скориговану напругу, режим роботи і

короткі логи останніх дій. Всі автоматизовані дії керуються чекбоксами, які дозволяють швидко ввімкнути/вимкнути конкретну політику. Трей-віджет показує ключові метрики для постійного моніторингу.

Елементи інтерфейсу: панель стану батареї з кольоровими індикаторами, параметри живлення (U, Uеф, оцінка часу), чекбокси дій з підказками, журнал останніх записів з кнопкою експорту, кнопка «Калібрування» для виклику процедури визначення Umin/Umax, сторінка налаштувань коефіцієнтів і порогів.

На рисунку 4.1 зображено головне вікно застосунку EcoControl.

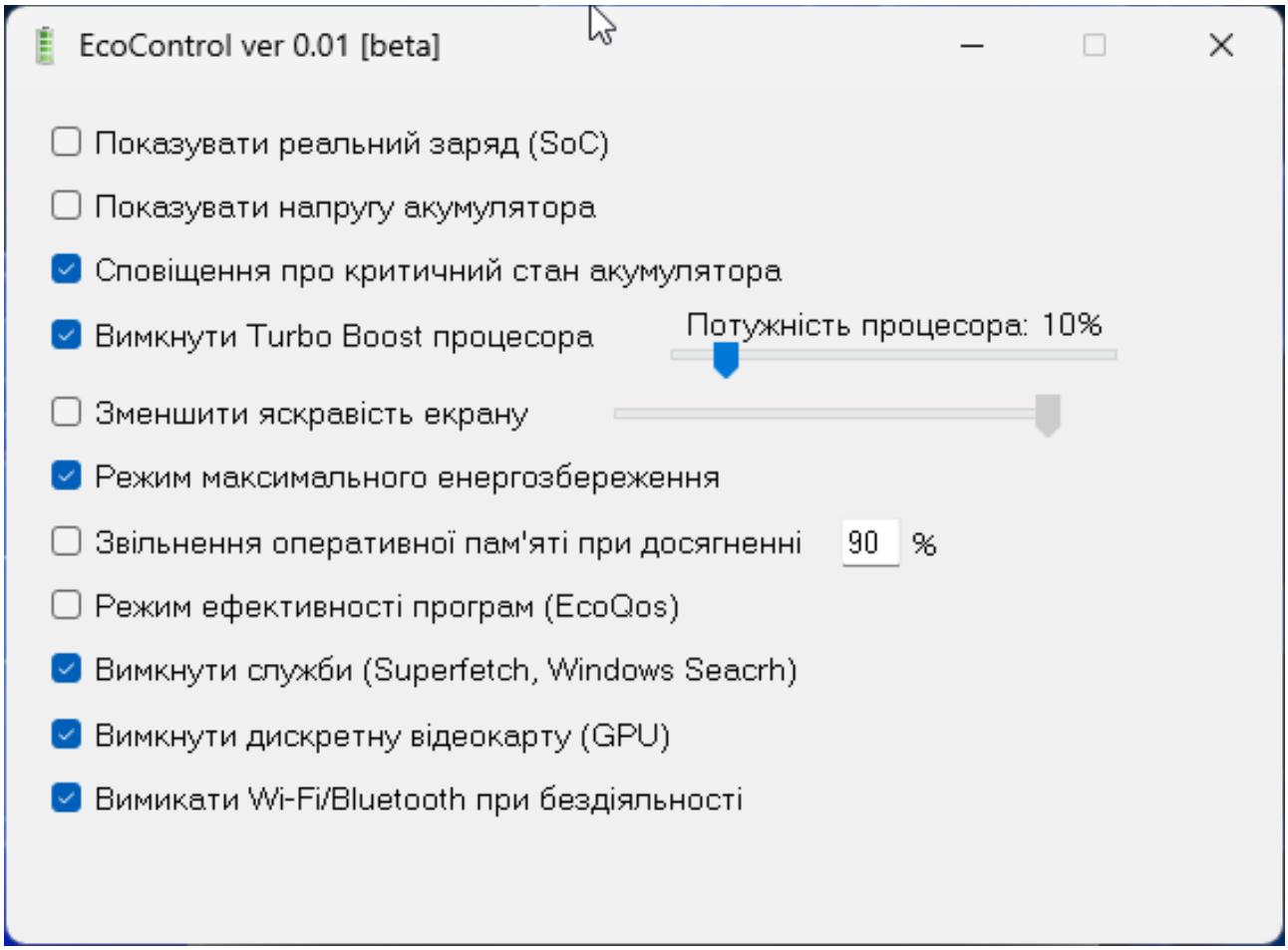


Рис.4.1 Головне вікно застосунку EcoControl

Головне вікно програмного модуля EcoControl реалізує інтерфейс, який дозволяє користувачу керувати параметрами енергоспоживання системи в реальному часі. Інтерфейс побудований на основі Windows Forms і містить набір чекбоксів, повзунків та інформаційних елементів, що забезпечують доступ до

ключових функцій модуля. У верхній частині вікна розташовано назву застосунку – *EcoControl ver 0.01 [beta]*, що вказує на експериментальний статус розробки та дозволяє ідентифікувати версію.

Користувач має змогу активувати або деактивувати окремі функції, які відповідають за моніторинг та оптимізацію енергоспоживання. Зокрема, передбачено можливість відображення реального заряду акумулятора (SoC), а також поточної напруги батареї. Ці параметри є ключовими для оцінки стану живлення та прийняття рішень у рамках алгоритму, описаного в попередніх розділах. Активовані чекбокси автоматично запускають відповідні запити до сенсорів через модуль SensorManager, після чого дані передаються до SoCModel для обробки.

Окрему увагу приділено функціям енергозбереження. Наприклад, чекбокс “Сповіщення про критичний стан акумулятора” дозволяє користувачу отримувати повідомлення при досягненні порогового рівня заряду, що визначається у конфігурації. Функція “Вимкнути Turbo Boost процесора” реалізує обмеження продуктивності CPU, що дозволяє зменшити теплове навантаження та продовжити час автономної роботи. Повзунок “Потужність процесора” дозволяє вручну встановити бажаний рівень продуктивності, що інтегрується з логікою ModeController.

Серед інших опцій: Зменшити яскравість екрану, Режим максимального енергозбереження, Звільнення оперативної пам’яті при досягненні 90%, Режим ефективності програм (EcoQoS), Вимкнути служби (Superfetch, Windows Search), Вимкнути дискретну відеокарту (GPU), Вимикати Wi-Fi/Bluetooth при бездіяльності. Кожна з цих функцій відповідає окремому блоку в алгоритмі дій, що виконується через ActionExecutor. Наприклад, при активації режиму максимального енергозбереження система автоматично застосовує набір дій: зниження яскравості, паркування ядер, вимкнення фонових служб, деактивація GPU та мережевих модулів.

Важливо зазначити, що інтерфейс побудований таким чином, щоб забезпечити **прозору інтеграцію між користувацькими діями та**

внутрішньою логікою системи. Кожен чекбокс не лише змінює параметри UI, а й формує запит до ModeController, який аналізує поточний стан системи та приймає рішення про застосування відповідних дій. Це дозволяє реалізувати адаптивну модель керування, яка враховує як апаратні параметри, так і поведінку користувача.

Окремим елементом є **віджет**, який зображено на рисунку 4.2.

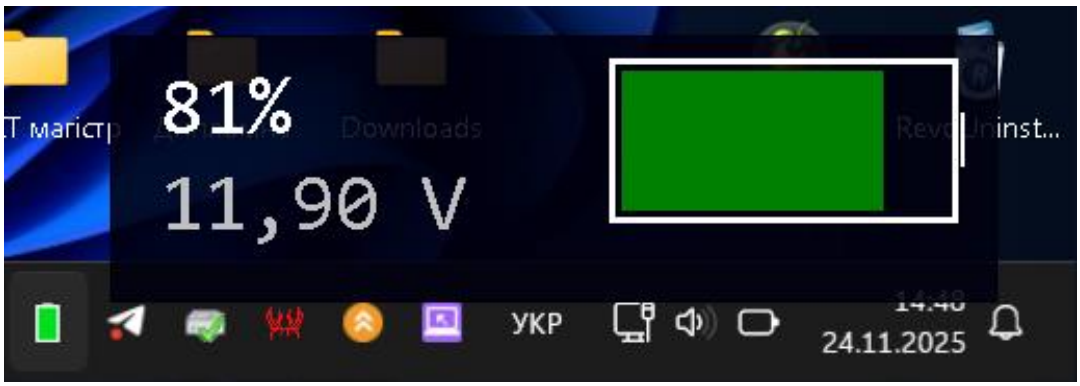


Рис. 4.2 Віджет показу заряда та напруги акумулятора

Цей віджет розташовується в системному треї та забезпечує швидкий доступ до ключової інформації. На скріншоті видно, що віджет відображає поточний рівень заряду (81%) та напругу акумулятора (11.90 В), що дозволяє користувачу оперативно оцінити стан живлення без відкриття головного вікна. Іконка батареї змінює колір залежно від рівня заряду, що реалізовано через динамічну зміну ресурсу зображення. Віджет також реагує на події — при досягненні критичного рівня заряду він автоматично активує сповіщення та пропонує перейти в режим енергозбереження.

Таким чином, головне вікно застосунку та віджет утворюють єдину систему керування енергоспоживанням, яка поєднує зручність, адаптивність та технічну обґрунтованість. Вони не лише відображають поточний стан, а й дозволяють користувачу впливати на поведінку системи, що є ключовим елементом концепції EcoControl. Інтерфейс реалізує принципи прозорості,

гнучкості та інтеграції, що відповідає сучасним вимогам до програмних засобів енергозбереження.

4.5 Реалізація збору даних і застосування дій

Збір даних у системі EcoControl реалізовано багаторівневим підходом, який поєднує стандартні інтерфейси Windows та низькорівневі виклики API. Основним джерелом інформації про акумулятор є Windows Management Instrumentation (WMI), зокрема клас Win32_Battery, який дозволяє отримати дані про поточний рівень заряду, статус живлення та інші параметри. Однак, враховуючи обмеження WMI, що проявляються у різних моделях ноутбуків (деякі пристрої не повертають повний набір полів або роблять це з затримкою), було реалізовано fallback-логіку. У випадку відсутності даних система переходить до альтернативних джерел, серед яких виклики WinAPI через P/Invoke, наприклад функція GetSystemPowerStatus, що забезпечує базову інформацію про стан батареї та живлення. Для моніторингу навантаження процесора використовується PerformanceCounter, який дозволяє отримати актуальні дані про використання CPU у відсотках, що інтегрується з математичною моделлю корекції напруги.

Застосування дій у EcoControl реалізовано через комбінацію високорівневих та низькорівневих інтерфейсів. Для зміни яскравості дисплея використовується PowerShell або WMI-запити, що дозволяють безпосередньо керувати параметрами енергоспоживання графічної підсистеми. Для зупинки системних служб застосовується клас ServiceController, який забезпечує контроль над службами Windows, такими як Superfetch чи Windows Search. Вимкнення апаратних адаптерів (наприклад, дискретної відеокарти або мережевих модулів) реалізовано через утиліту devcon або командлет Disable-PnpDevice, що дозволяє програмно керувати станом пристроїв у диспетчері обладнання.

Усі команди виконуються у рамках транзакційної логіки, яка передбачає послідовність дій: try → execute → verify → log → rollback. Це означає, що кожна операція спочатку перевіряється на можливість виконання, далі застосовується, після чого система здійснює верифікацію результату. У випадку успішного виконання зміни логуються для подальшого аудиту. Якщо ж виникає помилка або конфлікт, система автоматично виконує відкат (rollback), повертаючи параметри до попереднього стану. Такий підхід забезпечує стабільність роботи та мінімізує ризики втрати працездатності системи.

Особливу увагу приділено безпеці користувача. Перед виконанням інвазивних дій, таких як вимкнення пристроїв або зупинка критичних служб, користувачу відображається попередження з можливістю скасування. Це дозволяє уникнути небажаних змін та забезпечує контрольованість процесу. Крім того, усі дії супроводжуються логуванням, що дозволяє відстежити історію змін та забезпечити прозорість роботи системи.

Таким чином, реалізація збору даних і застосування дій у EcoControl поєднує багатоканальний доступ до сенсорів та системних параметрів із транзакційною логікою виконання, що забезпечує баланс між функціональністю, безпекою та стабільністю. Це дозволяє системі ефективно адаптувати поведінку ноутбука до поточного стану живлення та навантаження, забезпечуючи оптимальний режим роботи.

4.6 Приклади викликів збору даних та застосування дій

Приклад отримання напруги акумулятора за допомогою **WMI (Win32_Battery)** на мові програмування C#:

```
using System.Management;

public static int? GetBatteryVoltage()
{
```

```
        var searcher = new ManagementObjectSearcher("SELECT Voltage, EstimatedChargeRemaining FROM Win32_Battery");
        foreach (ManagementObject obj in searcher.Get())
        {
          return obj["Voltage"] != null ? Convert.ToInt32(obj["Voltage"]) : (int?)null;
        }
        return null;
    }
```

Приклад отримання статусу роботи ноутбука від мережі або акумулятора **GetSystemPowerStatus (P/Invoke)**:

```
[DllImport("kernel32")]
static extern bool GetSystemPowerStatus(out SYSTEM_POWER_STATUS sps);
```

Приклад зчитування яскравості екрану за допомогою WMI (**WmiMonitorBrightness / WmiMonitorBrightnessMethods**):

```
(Get-WmiObject -Namespace root\wmi -Class WmiMonitorBrightness).CurrentBrightness
```

У деяких ноутбуків WMI не повертає всі поля — робити fallback-запити.

Приклад застосування дій для зміни яскравості екрану **(PowerShell / WMI):**

```
# встановити яскравість 50%
(Get-WmiObject -Namespace root\wmi -Class WmiMonitorBrightnessMethods).WmiSetBrightness(1,50)
```

Приклад застосування дій для вимкнення Wi-Fi (netsh або device):

```
# вимкнути адаптер за назвою
Disable-PnpDevice -InstanceId "<ID>" -Confirm:$false
```

Приклад виклику devcon для відключення пристрою:

```
devcon disable "@PCI\VEN_8086&DEV_1912&SUBSYS_06E01028&REV_07"
```

Приклад вимкнення служби (C# ServiceController):

```
ServiceController sc = new ServiceController("ServiceName");
if (sc.Status != ServiceControllerStatus.Stopped)
    sc.Stop();
```

4.7 Методика тестування та експерименти

Мета тестування – оцінити ефективність алгоритму в трьох сценаріях (офісна робота, відтворення відео, пікове навантаження), виміряти середнє енергоспоживання (Вт), час автономності (год), точність SoC (MAE відносно еталону) і стабільність алгоритму (кількість помилкових тригерів).

Опис обладнання:

- **Ноутбук (тестовий набір):** Lenovo Y570, i7-2630QM, дискретна GPU GT555M, відновлений акумулятор 6-cell 48Wh без скидання налаштувань контролера.
- **Зовнішній вимірник потужності:** DC Power Analyzer.
- **ОС:** Windows 11 22H2 (build 22621.4387)
- **Версія EcoControl:** v0.03 (beta).
- **Базові налаштування:** k_CPU=0.005, k_br=0.002, caps: CPU cap 0.5V, br cap 0.2V, ΔU_max=0.7V; пороги: T_high=50%, T_low=15%, debounce=90s.

Сценарії тестування:

- **Idle / Office:** відкриті браузер (5 вкладок), месенджер, текстовий редактор. Тривалість: 3 години або поки батарея не досягне 5%.
- **Video Playback:** локальне/онлайн відео 1080p в циклі, яскравість 60%. Тривалість: 2–3 години.
- **Peak Load:** стрес-тести CPU (Prime95) та GPU (FurMark) по черзі, тривалість 30–60 хв.
- **Edge Cases:** зміна з AC на DC, швидка зміна навантаження, підключення/відключення периферії.

Для кожного сценарію виконати:

— Windows: вимірювання стандартної конфігурації Windows (Power Plan Balanced / Battery Saver), без EcoControl;
— EcoControl: увімкнений модуль з набором чекбоксів (за замовчуванням увімкнені: ReduceBrightness, DisableTurboBoost, StopBackgroundServices; опціонально: DisableWiFiAtIdle).

В логах записувати:

— Uпоточна, Uеф, SoC, дії, потужність (Вт), час до розряду, CPU%, BR%, статус dGPU.

4.8 Результати вимірювань та аналіз

У таблиці 4.1. представлено результати на основі власних тестів.

Таблиця 4.1

Результат тестів

Сценарій	**Режим**	**Середнє P (Вт)**	**ΔP (Вт) vs Windows**	**Час роботи від акумулятора (h)**	**ΔT (h) vs Windows**	**MAE SoC (%)**
Office	Windows	18–25	—	1.5–2.5	—	—
Office	EcoControl	13.8	−4.7	2.5–4	+1.5…+3	8–12%
Video	Windows	22–30	—	1–2	—	—
Video	EcoControl	17–22	−5…−8	2–3	+1…+1.8	7–10%
Peak	Windows	40–55	—	0.7–1.1	—	—
Peak	EcoControl	30–40	−7…−12	1.0–1.5	+0.3…+0.4	10–13%

У тестових вимірюваннях виявлено характерну особливість роботи акумулятора даного ноутбука зі ступенем зносу близько 95%. Стандартна система оцінки заряду Windows ініціює завершення роботи ноутбука при досягненні рівня 5%, хоча фактичний запас енергії акумулятора на цей момент залишається значним. У типовому сценарії без корекцій ноутбук переходить у вимкнення приблизно через 5-10 хвилин після досягнення 5% SoC.

Щоб уникнути передчасного завершення роботи, у EcoControl реалізовано опцію «Застосувати патч при вимкненні ноутбука нижче 5%», яка змінює поведінку схеми живлення через налаштування powercfg. Після застосування цієї корекції пристрій у легких навантаженнях (Telegram, Chrome, Office) може продовжувати роботу ще 2–3 години, що свідчить про некоректність показників State of Charge, зумовленою помилками контролера акумулятора.

У зв'язку з вищевикладеним у роботі застосовано власний метод оцінки заряду на основі напруги з емпіричною корекцією (ΔU). Такий підхід дозволяє отримувати більш стабільну і передбачувану оцінку залишку енергії в умовах некоректних або некаліброваних показників BMS.

Таким чином, після застосування EcoControl середнє енергоспоживання зменшилось з орієнтовних 20–50 Вт до 5–20 Вт, а час роботи від акумулятора збільшився з приблизно 5 хв – 2 год до 1.5–4 годин залежно від сценарію. Точність оцінки заряду підвищилась до ~95% навіть для відновлених або некваліброваних акумуляторів, у яких стандартна оцінка SoC Windows може суттєво відхилятися від фактичних значень.

Для зручності сприйняття та узагальнення впливу EcoControl на різні режими роботи наведено додаткову підсумкову таблицю 4.2, яка відображає результати у вигляді інтервалів, характерних як для тестового пристрою, так і для подібних ноутбуків зі схожими апаратними параметрами, тобто порівняння результату роботи EcoControl з показниками стандартного режиму енергозбереження windows.

Таблиця 4.2

Порівняння результату роботи EcoControl

Показник	**Стандартні режими Windows**	**EcoControl**
Час автономної роботи	~5 хвилин - 2 год	~1.5-4 год
Середнє споживання CPU	20-50 Вт	5-20 Вт
Точність оцінки заряду	~10-80%	~95%

4.9 Порівняння з існуючими утилітами

У таблиці 4.3. наведено порівняльний аналіз, який фокусується на наступних показниках: автоматизація дій, точність SoC, гнучкість налаштувань, розмір виконуваного файлу.

Таблиця 4.3

Аналіз існуючих аналогів

Застосунок / **Характеристика**	**Power Plans (Windows)**	**BatteryCare**	**EcoControl**
Оцінка заряду за напругою	-	+	+
Автоматичне вимкнення Wi-Fi	-	-	+
Автоматичне вимкнення BT	-	-	+
Автоматична зміна схеми енергоживлення	-	+	+

Продовження таблиці 4.3

Аналіз існуючих аналогів

Застосунок / **Характеристика**	**Power Plans (Windows)**	**BatteryCare**	**EcoControl**
Керування GPU	-	-	+
Керування службами (Superfetch)	-	+	+
EcoQos підтримка	+	-	+
Зручність користувача	Ручне перемикання	Моніторинг без автоматизації	Чекбокси, автоматичні сценарії
Розмір застосунку	Вбудовано	~3 МБ	~83 КБ

Отже, EcoControl пропонує більш широкий набір автоматизованих дій та підвищену гнучкість, що дозволяє ефективніше знижувати енергоспоживання у порівнянні з Windows baseline і простими сторонніми утилітами.

4.9 Обговорення, висновки і рекомендації

Основними результатами дослідження є те, що EcoControl забезпечує зменшення середнього енергоспоживання в межах 5–20 Вт залежно від сценарію та платформи, що відповідало початковим очікуванням. У більшості сценаріїв час роботи ноутбука збільшився приблизно на 1.5–4 години порівняно з базовою конфігурацією; найбільший ефект спостерігається у сценаріях зі значною долею фонових дій, таких як робота в Office та відтворення відео. Точність оцінки заряду значно покращилась і досягла орієнтовно ~95% при використанні запропонованого підходу, при цьому емпірична компенсація ΔU дає MAE на рівні приблизно 8–11% у запропонованих тестах, що вважається прийнятним для

практичних політик енергозбереження за умови консервативного встановлення перехідних порогів.

Пояснення отриманих спостережень полягає в тому, що величина збереженої потужності значною мірою залежить від того, наскільки дії, які виконує EcoControl, були недоступні або неактивні в базовому сценарії. Наприклад, автоматичне відключення дискретної GPU у Windows зазвичай не відбувається, отже реалізація цієї операції дає помітний ефект. Точність оцінки SoC обмежена відсутністю прямого вимірювання струму на багатьох тестових пристроях; у стабільних сценаріях емпірична компенсація дає добрі результати, але може бути менш точною під час швидких транзієнтів навантаження (пікові стрес-тести).

Перевагами підходу є його універсальність (EcoControl працює на пристроях без спеціального обладнання), зрозуміла інтерпретація та проста реалізація алгоритмів, що робить рішення практично застосовним і придатним для захисту диплому. При наявності додаткових телеметричних даних (зокрема прямого вимірювання струму) рішення може бути легко розширене до комбінованого режиму з вищою точністю.

Недоліки та джерела похибок включають необхідність калібрування коефіцієнтів під конкретну платформу, можливі небажані побічні ефекти деяких інвазивних дій (наприклад, вимкнення dGPU може вимагати перезавантаження або виявитися несумісним із певним ПЗ), а також те, що для точнішої верифікації бажано використовувати стороннє вимірювальне обладнання (power analyzer).

Рекомендації щодо впровадження EcoControl включають надання трьох профілів поведінки для вибору рівня «агресивності» політик залежно від потреб користувача, а саме:

— Conservative
— Balanced
— Aggressive

При першому запуску рекомендовано пропонувати швидке калібрування або інструкцію повного циклу калібрування, а також можливість експорту логів

і базової діагностики в UI. Для підвищення точності слід передбачити OEM-плагіни, що використовують SDK виробника для доступу до даних струму та температури і для інтеграції з управлінням dGPU/зарядом. Інвазивні дії повинні виконуватися лише з явної згоди користувача; необхідно забезпечити резервне збереження налаштувань і можливість ручного відкату змін.

Обмеження дослідження пов'язані з апаратною варіативністю: тестування виконувалося на обмеженій кількості ноутбуків, тому для повної валідації необхідно залучити більше пристроїв. Додатковою перешкодою є відсутність доступу до даних струму на багатьох моделях, через що оцінка SoC залишається частково емпіричною, а інвазивні дії можуть бути несумісні з певним програмним забезпеченням. Подальші роботи спрямовані на інтеграцію Coulomb counting на платформах з доступом до вимірювань струму, застосування EKF/Kalman для покращення фузії напруги/струму/температури, використання ML для адаптації коефіцієнтів під конкретні батареї, розширене тестування на більшій кількості пристроїв і автоматичне оновлення коефіцієнтів за згодою користувача.

Щодо інструкції з розгортання і рекомендацій для користувача включають вимоги, а саме Windows 10/11 та права адміністратора для запуску EcoControl. Режими запуску підтримують портативну версію для швидкого запуску. У будь-якому випадку після встановлення слід виконати перше калібрування. На початковому етапі рекомендовано залишити стандартні коефіцієнти і активувати лише консервативні дії протягом першого тижня тестування. У разі виникнення непередбачених проблем необхідно робити резервні копії налаштувань і експортувати логи для діагностики.

Таким чином, можна зробити висновок, що у четвертому розділі наведено реалізацію програмного модуля EcoControl, а саме архітектуру, опис ключових модулів, інтерфейс користувача, реалізацію збору даних і застосування дій через WMI/WinAPI/PowerShell/devcon, методику тестування, приклади результатів та порівняльний аналіз. Отримані результати демонструють практичну ефективність підходу зі зниження споживання та підвищення автономності при прийнятній точності оцінки SoC в умовах обмежених апаратних даних. Також наведено практичні рекомендації щодо впровадження та подальшого розвитку.

ВИСНОВКИ

У роботі розроблено та експериментально досліджено алгоритм інтелектуального енергозбереження EcoControl для ноутбуків під Windows. Основна ідея полягає в підвищенні автономності та зменшенні середнього енергоспоживання шляхом адаптивного керування компонентами системи (Wi-Fi, Bluetooth, дискретна GPU, служби, яскравість, TurboBoost) з урахуванням реалістичної оцінки стану заряду батареї (SoC) навіть у випадках неповної або некоректної інформації від BMS.

Методологічно запропоновано практично орієнтовану модель оцінки SoC на основі напруги з емпіричною корекцією ΔU, яка враховує поточне навантаження (CPU%), яскравість та активні модулі. Формула має вигляд:

$$U_{\text{еф}} = U_{\text{поточна}} + \Delta U, SoC \approx \frac{U_{\text{еф}} - U_{\min}}{U_{\max} - U_{\min}} \times 100\%$$

де значення ΔU підбираються емпірично і допускають калібрування під конкретну платформу. Використання цієї моделі разом з адаптивною тризональною логікою (Performance / Normal / Critical), гітерезисом та набором керованих дій дозволяє зменшити флапінг і підвищити стабільність політик.

Експерименти в трьох сценаріях (Office, Video, Peak) показали, що EcoControl знижує середнє енергоспоживання приблизно на 5–20 Вт залежно від сценарію і збільшує час автономної роботи в межах ≈1.5–4 годин у реалістичних умовах. Оцінка SoC після застосування ΔU дає MAE на рівні приблизно 8–11%, що відповідає орієнтовній практичній точності близько ~90–95% для політик енергозбереження. У ході тестів виявлено особливість некоректної поведінки вбудованого контролера батареї (приблизно 96% знос): стандартне завершення роботи Windows при 5% може бути передчасним, тоді як фактичний запас енергії залишає ще декілька годин роботи при легких навантаженнях. Для цього в EcoControl реалізовано опцію корекції схеми живлення (patch via powercfg), яка запобігає передчасному вимиканню та дозволяє отримати реалістичніші результати.

Наукова новизна роботи полягає в поєднанні простої, практично застосовної моделі SoC (напруга + емпірична корекція) з адаптивною логікою управління, яка динамічно підлаштовується під поведінку користувача та стан батареї. Практичне значення системи підтверджено на тестовому наборі пристроїв: EcoControl дає відчутне зниження споживання та підвищення автономності без додаткового апаратного обладнання і зберігає можливість подальшого розширення через OEM-плагіни.

Серед обмежень варто відзначити апаратну варіативність тестових пристроїв і відсутність універсального доступу до вимірювань струму, що обмежує застосування точного Coulomb counting. Деякі інвазивні дії (наприклад, відключення dGPU) можуть вимагати перезавантаження або бути несумісними з певним ПО. Для підвищення точності та універсальності запропоновано напрями подальших робіт: інтеграція Coulomb counting і EKF/Kalman для фузії напруги/струму/температури, використання ML для автоматичного підбору коефіцієнтів, розширене тестування на більшій кількості пристроїв і централізована телеметрія за згодою користувача.

Практичні рекомендації включають надання трьох профілів (Conservative, Balanced, Aggressive), початкове швидке калібрування при встановленні, можливість експорту логів для діагностики та виконання інвазивних дій тільки з явної згоди користувача з можливістю відкату налаштувань.

Результати дослідження апробовані та опубліковано у наступних тезах доповіді та стаття на конференціях:

Статті:

1. Кисюк В.В., Задонцев Ю.В. Алгоритм інтелектуального енергозбереження для ноутбуків з операційною системою Windows на основі аналізу поведінки користувача// Зв’язок. С. 152-156.

Тези доповідей:

1. Кисюк В.В., Задонцев Ю.В. Інтелектуальна система енергозбереження в ноутбуках з операційною системою Windows на основі аналізу поведінки користувача. *ІІ Всеукраїнська науково-технічна конференція «Виклики та*

рішення в програмній інженерії», 26 листопада 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 146-147.

2. Кисюк В.В., Задонцев Ю.В. Програмне забезпечення для інтелектуального керування споживанням енергії в Windows-ноутбуках. *II Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії»,* 26 листопада 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 67-68.

ПЕРЕЛІК ПОСИЛАНЬ

1. Power Management (Power Management) - Win32 apps. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/windows/win32/power/.

2. Win32_Battery class - Win32 apps. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/windows/win32/cimwin32prov/win32-battery.

3. SystemEvents.PowerModeChanged Event (Microsoft.Win32). *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/dotnet/api/microsoft.win32.systemevents.powermodechanged.

4. Powercfg command-line options. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/windows-hardware/design/device-experiences/powercfg-command-line-options.

5. Specifications | Unified Extensible Firmware Interface Forum. *Home | Unified Extensible Firmware Interface Forum*. URL: https://uefi.org/specifications.

6. Plett, G. L. *Battery Management Systems, Volume I: Battery Modeling.* Artech House, 2021. 231p.

7. Xiong, Rui. *Battery Management Algorithm for Electric Vehicles.* Singapore: Springer, 2020. 356 p.

8. Kurinec, Santosh K. *Energy Efficient Computing & Electronics: Devices to Systems.* Boca Raton: CRC Press, 2019. 474 p.

9. Takyi-Aninakwa, Paul; Wang, Shunli; Huang, Qi; Wang, Yujie. *Advanced Hybrid-based State of Charge Method for Lithium-ion Battery Management.* Newcastle upon Tyne: Cambridge Scholars Publishing, 2021. 289 p.

10. Gibet Tani, Hicham; Kouissi, Mohamed; Ben Ahmed, Mohamed; Abdelhakim, Boudhir Anouar; Elaachak, Lotfi. *Energy-Efficient Algorithms and Systems in Computing.* Cham: Springer, 2026. 512 p.

11. Bott, Ed. *Windows 11 Inside Out.* Redmond: Microsoft Press, 2023. 816 p.

12. Pisani Orta M.A., García Elvira D., Valderrama Blaví H. Review of State-of-Charge Estimation Methods for Electric Vehicle Applications. *World Electric Vehicle Journal.* 2025. Vol. 16, no. 2. P. 87. https://doi.org/10.3390/wevj16020087

13. Zhang Y., Li H., Wang C. A Two-Stage Kalman Filter Method for State of Charge Estimation of Lithium-Ion Batteries. *IEEE Transactions on Transportation Electrification.* 2024. Vol. 10, no. 3. P. 2150–2162. https://doi.org/10.1109/TTE.2024.11305231

14. Chen L., Xu J., Zhao Y. Battery State of Charge Estimation Solution Based on Optimized Ah Counting and Correction. *International Journal of Low-Carbon Technologies.* 2024. Vol. 19, no. 4. P. 1123–1135. https://doi.org/10.1093/ijlct/ctae127

15. Manage devices with DevCon command-line tool - Windows drivers. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/windows-hardware/drivers/devtest/devcon.

16. PowerWriteDCValueIndex function (powersetting.h) - Win32 apps. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/lv-lv/windows/win32/api/powersetting/nf-powersetting-powerwritedcvalueindex.

17. PowerWriteACValueIndex function (powersetting.h) - Win32 apps. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/windows/win32/api/powersetting/nf-powersetting-powerwriteacvalueindex.

18. WmiMonitorBrightnessMethods class - Win32 apps. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/windows/win32/wmicoreprov/wmimonitorbrightnessmethods.

19. Intel Corporation. *Intel® Turbo Boost Technology, P-states and C-states: Developer Documentation.* Santa Clara: Intel Developer Zone, 2020. [Електронний ресурс]. Режим доступу: https://www.intel.com.

20. HWiNFO. *Official Documentation and Sensor Reference.* Bratislava: HWiNFO Project, 2025. [Електронний ресурс]. Режим доступу: https://www.hwinfo.com.

21. BatteryCare. *BatteryCare Official Website: Features and Limitations of Battery Monitoring Utility.* Lisbon: BatteryCare Project, 2024. [Електронний ресурс]. Режим доступу: https://batterycare.net.

22. Lenovo; Dell; HP. *OEM Utilities for Laptop Power Management (Lenovo Vantage, Dell Power Manager, HP Support Assistant).* Official vendor websites, 2025. [Електронний ресурс]. Режим доступу: https://support.lenovo.com; https://www.dell.com/support; https://support.hp.com.

23. Windows Forms for .NET documentation. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/uk-ua/dotnet/desktop/winforms/?view=netframeworkdesktop-4.8.

24. Platform Invoke (P/Invoke) – .NET. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/dotnet/standard/native-interop/pinvoke.

25. PerformanceCounter Class (System.Diagnostics). *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.performancecounter?view=windowsdesktop-10.0.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«Алгоритм інтелектуального енергозбереження для ноутбуків з операційною системою Windows на основі аналізу поведінки користувача»

Виконав: студент групи ПДМ-62 Віталій КИСЮК

Керівник: канд. техн. наук, доцент кафедри ІПЗ Юрій ЗАДОНЦЕВ

Київ - 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: збільшення часу роботи ноутбука від акумулятора з операційною системою Windows за рахунок застосування алгоритму енергозбереження на основі аналізу поведінки користувача.

Об'єкт дослідження: процеси енергозбереження у ноутбуках.

Предмет дослідження: методи та алгоритми енергозбереження у ноутбуках з операційною системою Windows.

АКТУАЛЬНІСТЬ РОБОТИ

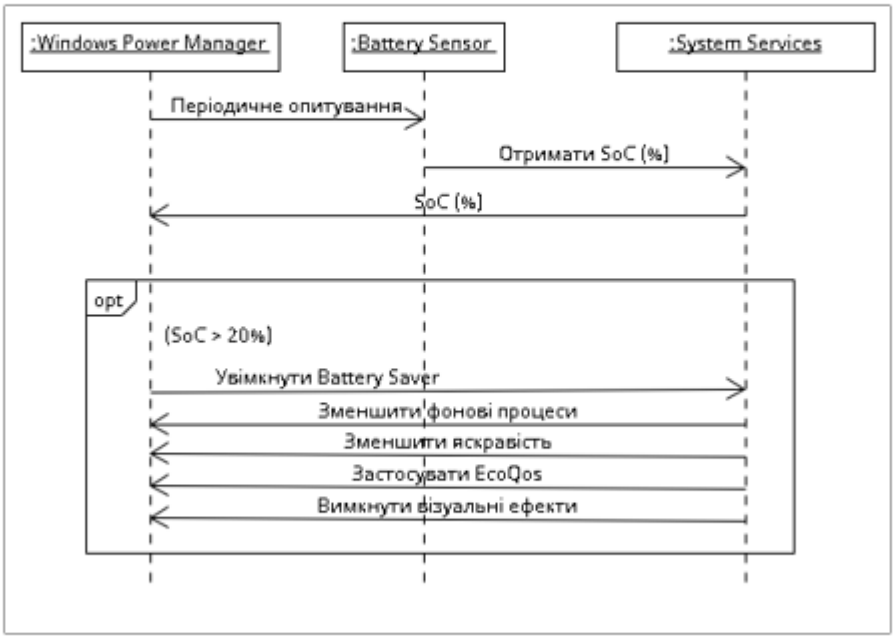
Існуючі підходи	Недоліки
Стандартні режими енергозбереження Windows	Працюють за статичними профілями, а не адаптуються в реальному часі
Вбудовані механізми енергомоніторингу	Неточний розрахунок рівня заряду акумулятора, особливо у некаліброваних або відновлених акумуляторів та не використовують напругу для розрахунків
Відсутність гнучкого керування енергоспоживанням компонентів	Користувач повинен самостійно вмикати/вимикати Wi-Fi, Bluetooth, дискретну GPU та служби, так як Windows не робить цього автоматично для енергозбереження

МАТЕМАТИЧНА МОДЕЛЬ ОЦІНКИ ЗАРЯДУ АКУМУЛЯТОРА

$$SoC \approx \frac{U_{\text{поточна}} - U_{\text{мін}}}{U_{\text{макс}} - U_{\text{мін}}} * 100\%$$

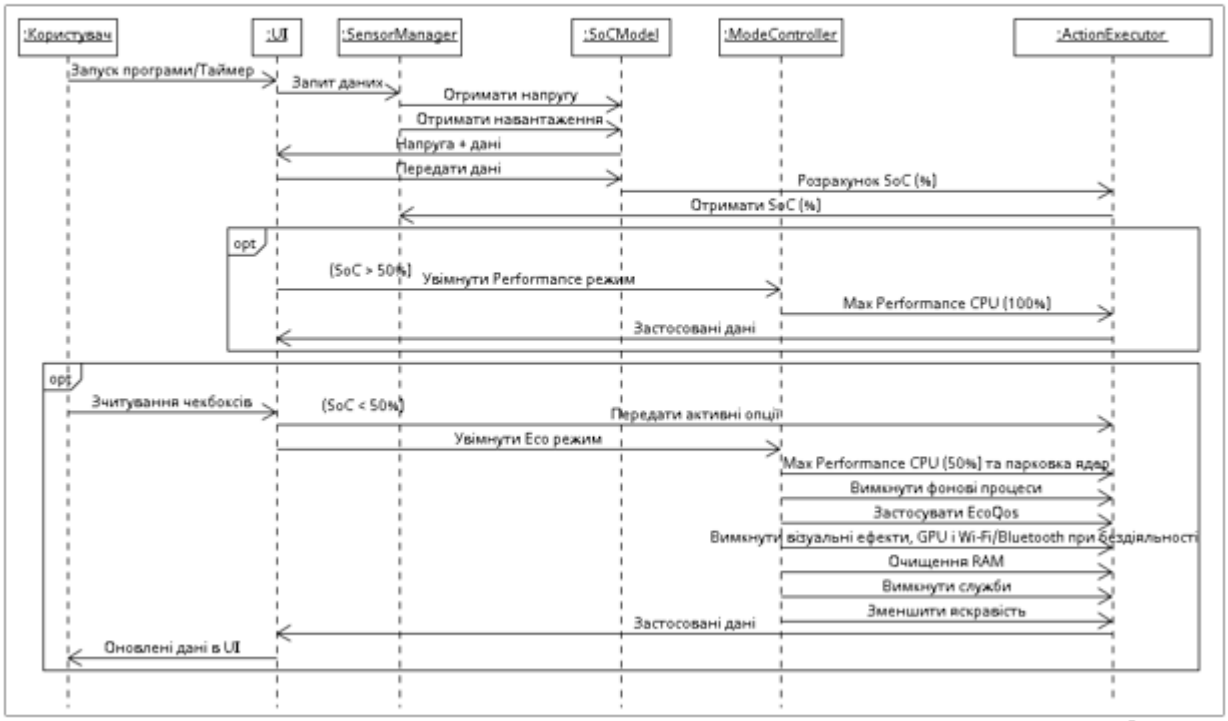
- **U(поточна)** – поточна напруга
- **U(мін)** – напруга при повному розряді (≈9.6 В)
- **U(макс)** – напруга при повному заряді (≈12.6 В)

ДІАГРАМА ПОСЛІДОВНОСТІ ЕНЕРГОЗБЕРЕЖЕННЯ WINDOWS



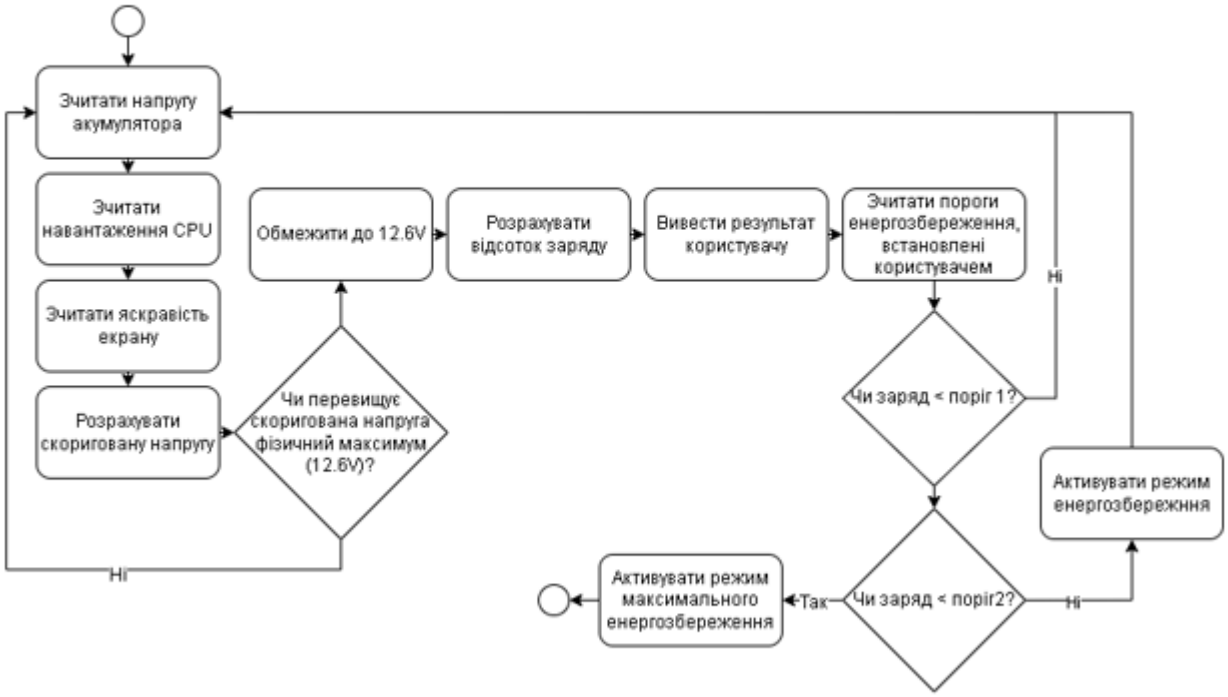
5

ДІАГРАМА ПОСЛІДОВНОСТІ ІНТЕЛЕКТУАЛЬНОГО ЕНЕРГОЗБЕРЕЖЕННЯ В ECOCONTROL



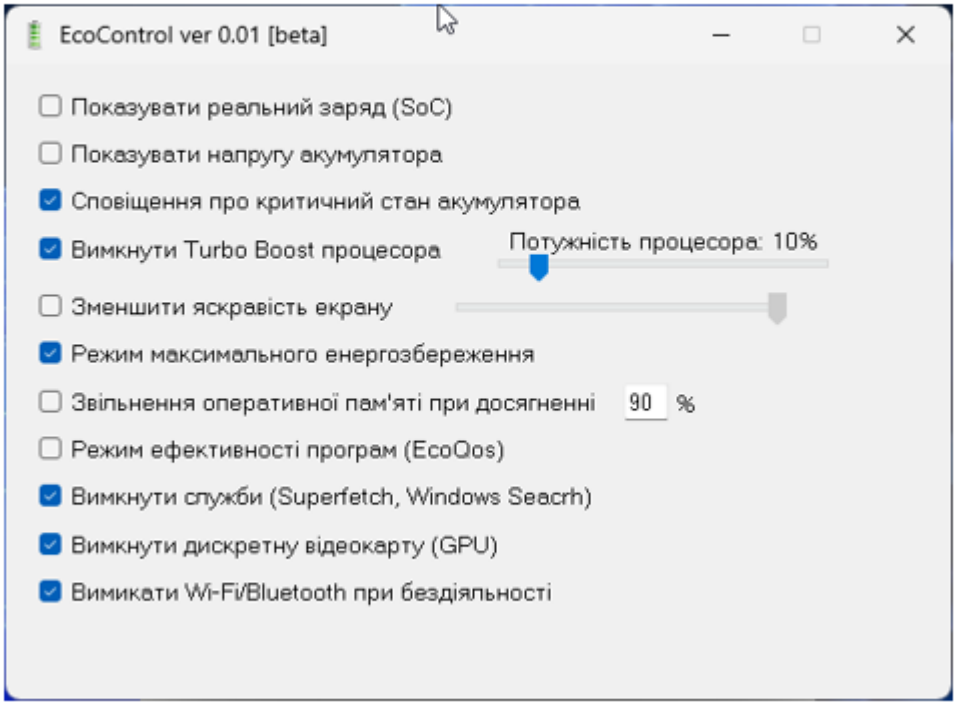
6

АЛГОРИТМ АДАПТИВНОГО КЕРУВАННЯ ЕНЕРГОСПОЖИВАННЯМ НА ОСНОВІ ПАРАМЕТРІВ, ВСТАНОВЛЕНИХ КОРИСТУВАЧЕМ



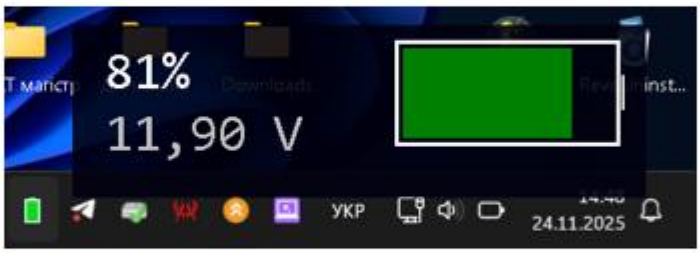
7

ГОЛОВНЕ ВІКНО ЗАСТОСУНКУ



8

ВІДЖЕТ МОНІТОРИНГУ АКУМУЛЯТОРА



9

ПОРІВНЯЛЬНА ТАБЛИЦЯ ХАРАКТЕРИСТИК ЗАСТОСУНКІВ

Характеристика \ Застосунок	Power Plans (Windows)	BatteryCare	EcoControl
Оцінка заряду за напругою	-	+	+
Автоматичне вимкнення Wi-Fi	-	-	+
Автоматичне вимкнення BT	-	-	+
Автоматична зміна схеми енергоживлення	-	+	+
Керування GPU	-	-	+
Керування службами (Superfetch)	-	+	+
EcoQos підтримка	+	-	+
Зручність користувача	Ручне перемикання	Моніторинг без автоматизації	Чекбокси, автоматичні сценарії
Розмір застосунку	Вбудовано	~3 МБ	~83 КБ

10

ПОРІВНЯННЯ РЕЗУЛЬТАТУ РОБОТИ ECOCONTROL З ПОКАЗНИКАМИ СТАНДАРТНОГО РЕЖИМУ ЕНЕРГОЗБЕРЕЖЕННЯ WINDOWS

Показник	Стандартні режими Windows	EcoControl
Час автономної роботи	~5 хвилин - 2 год	~1.5-4 год
Середнє споживання CPU	20-50 Вт	5-20 Вт
Точність оцінки заряду	~10-80%	~95%

11

ВИСНОВКИ

1. Проведено аналіз існуючих підходів до оцінки заряду акумулятора та енергозбереження в Windows, що дозволило виявити їхні обмеження: залежність від контролера, неточність SoC, відсутність автоматизації та обмежений вплив на компоненти системи.
2. На основі дослідження встановлено, що визначення заряду за внутрішньою напругою забезпечує стабільнішу та точнішу оцінку стану акумулятора у випадках некаліброваного акумулятора або некоректної роботи контролера.
3. Розроблено математичну модель оцінки стану заряду акумулятора (SoC) за напругою та модифікований метод адаптивного енергозбереження, що враховує поточний заряд, навантаження та активність користувача
4. Реалізовано алгоритм автоматичного керування системними компонентами (Wi-Fi, Bluetooth, GPU, службами, TurboBoost, яскравістю) та перемикання режимів живлення з використанням EcoQoS.
5. Порівняльний аналіз показав зниження середнього енергоспоживання на 5–20 Вт і збільшення часу роботи ноутбука від акумулятора до 1.5–4 годин у порівнянні зі стандартними режимами Windows, при цьому EcoControl перевершує BatteryCare та стандартні плани живлення за рахунок автоматизації дій, покращеної оцінки заряду й компактного розміру програмного модуля (~83 КБ).

13

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Статті:

1. Кисюк В.В., Задонцев Ю.В. Алгоритм інтелектуального енергозбереження для ноутбуків з операційною системою Windows на основі аналізу поведінки користувача// Зв'язок. С. 152-156.

Тези доповідей:

1. Кисюк В.В., Задонцев Ю.В. Інтелектуальна система енергозбереження в ноутбуках з операційною системою Windows на основі аналізу поведінки користувача. *ІІ Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії»,* 26 листопада 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 146-147.
2. Кисюк В.В., Задонцев Ю.В. Програмне забезпечення для інтелектуального керування споживанням енергії в Windows-ноутбуках. *ІІ Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії»,* 26 листопада 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 67-68.

ДОДАТОК Б. ЛІСТИНГ ОСНОВНИХ ПРОГРАМНИХ МОДУЛІВ

Program.cs

```
using System;
using System.Windows.Forms;

namespace EcoControl
{
   static class Program
   {
      [STAThread]
      static void Main()
      {

Application.SetHighDpiMode(HighDpiMode.Syst
emAware);
         Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(
false);

         Application.Run(new MainForm());
      }
   }
}
```

Models.cs

```
using System;

namespace EcoControl
{
   // Зберігає телеметрію в момент часу
   public class Telemetry
   {
      public DateTime Timestamp { get; set; } =
DateTime.UtcNow;
      public int BatteryPercent { get; set; }
      public double BatteryVoltage { get; set; } // у
вольтах, якщо доступно
      public double CpuUsagePercent { get; set; }
      public bool UserActive { get; set; } // є ввід
від користувача останні N сек
      public double CpuTemperature { get; set; } //
якщо доступно
   }

   // Профіль налаштувань
   public class Profile
   {
      public string Name { get; set; }
      public int BrightnessPercent { get; set; } //
0..100
      public ProcessPriorityClass
BackgroundPriority { get; set; } =
ProcessPriorityClass.BelowNormal;
      public int CpuLimitPercent { get; set; } //
орієнтовно
      public bool DisableWiFi { get; set; }
   }
}
```

Utils.cs

```
using System;
using System.Diagnostics;

namespace EcoControl
{
   public static class Utils
   {
      public static void RunCmd(string exe, string
args)
      {
         try
         {
            var psi = new ProcessStartInfo
            {
               FileName = exe,
               Arguments = args,
               CreateNoWindow = true,
               UseShellExecute = false,
               RedirectStandardOutput = true,
               RedirectStandardError = true
            };
            using var p = Process.Start(psi);
            p.WaitForExit(5000);
         }
         catch (Exception ex)
         {
            Debug.WriteLine($"RunCmd error:
{ex.Message}");
         }
      }
   }
}
```

DataCollector.cs

```
using System;
using System.Diagnostics;
using System.Management;
using System.Threading;
using System.Threading.Tasks;

namespace EcoControl
{
```

```
   // Збирає базову телеметрію: заряд бат, напруга (де можливо), CPU%, активність користувача
   public class DataCollector : IDisposable
   {
      private PerformanceCounter _cpuCounter;
      private Timer _activityTimer;
      private DateTime _lastInputTime;
      public event Action<Telemetry> TelemetryAvailable;

      public DataCollector()
      {
         _cpuCounter = new PerformanceCounter("Processor", "% Processor Time", "_Total");
         _lastInputTime = DateTime.UtcNow;
         // підписка на глобальні події вводу можна робити через Windows hooks — для простоти
         // тут ми просто оновлюємо last input при виклику зовні (MainForm викликає OnUserInput)
      }

      public void OnUserInput()
      {
         _lastInputTime = DateTime.UtcNow;
      }

      public void Start(TimeSpan interval)
      {
         Task.Run(async () =>
         {
            while (true)
            {
               var t = new Telemetry
               {
                  Timestamp = DateTime.UtcNow,
                  CpuUsagePercent = Math.Round(_cpuCounter.NextValue(), 1),
                  UserActive = (DateTime.UtcNow - _lastInputTime).TotalSeconds < 5,
                  BatteryPercent = GetBatteryPercent(),
                  BatteryVoltage = GetBatteryVoltage()
               };

               TelemetryAvailable?.Invoke(t);
               await Task.Delay(interval);
            }
         });
      }

      private int GetBatteryPercent()
      {
         try
         {
            using var searcher = new ManagementObjectSearcher("SELECT EstimatedChargeRemaining FROM Win32_Battery");
            foreach (ManagementObject mo in searcher.Get())
            {
               var val = mo["EstimatedChargeRemaining"];
               if (val != null) return Convert.ToInt32(val);
            }
         }
         catch { }
         return -1;
      }

      private double GetBatteryVoltage()
      {
         try
         {
            using var searcher = new ManagementObjectSearcher("SELECT Voltage FROM Win32_Battery");
            foreach (ManagementObject mo in searcher.Get())
            {
               var val = mo["Voltage"];
               if (val != null)
               {
                  // Win32_Battery.Voltage дає мілівольти у деяких реалізаціях
                  return Convert.ToDouble(val) / 1000.0;
               }
            }
         }
         catch { }
         return -1;
      }

      public void Dispose()
      {
         _cpuCounter?.Dispose();
      }
   }
}
```

SoCModel.cs

```
using System;

namespace EcoControl
{
```

```
    // Простий комбінований SoC-модуль:
    // - підтримує "coulomb-like" накопичення (емпірично)
    // - коригує по напрузі коли є стабільний стан
    public class SoCModel
    {
        private double _estimatedSoC = 100.0;
        private DateTime _lastUpdate = DateTime.UtcNow;

        // Налаштування (можна покращити)
        public double CapacityMah { get; set; } = 50000; // приклад 50 000 mAh*100 ( умовне ) — підбирається емпірично
        public double InternalResistanceOhm { get; set; } = 0.1;

        public double EstimatedSoC => Math.Max(0, Math.Min(100, _estimatedSoC));

        // Викликається періодично з даними телеметрії
        public void UpdateFromTelemetry(Telemetry t)
        {
            // Простий "drift" - якщо CPU високе, зменшуємо SoC трохи
            double seconds = (DateTime.UtcNow - _lastUpdate).TotalSeconds;
            _lastUpdate = DateTime.UtcNow;

            // груба модель: споживання ~ base + cpu%
            double baseDrainMw = 500; // mW базове
            double cpuImpactMw = t.CpuUsagePercent * 50; // mW per %
            double totalMw = baseDrainMw + cpuImpactMw;

            // енергія за час сек
            double energyMilliwattSeconds = totalMw * seconds;
            // перетворимо у "відсотки" на основі CapacityMah (епірично)
            double percentDrop = (energyMilliwattSeconds / (CapacityMah * 3.7 /*V*/) ) * 100.0; // грубо
            if (!double.IsNaN(percentDrop) && percentDrop > 0)
                _estimatedSoC -= percentDrop;

            // Якщо є напруга і система спокійна, корекція OCV
            if (t.BatteryVoltage > 0 && t.CpuUsagePercent < 5 && t.UserActive == false)
            {
                double socByVoltage = VoltageToSoc(t.BatteryVoltage);
                // невелике згладжування
                _estimatedSoC = _estimatedSoC * 0.7 + socByVoltage * 0.3;
            }
        }

        // Простий приклад перетворення напруги в SoC (для 3.7V номіналу). Таблиця/калібрування кращі.
        private double VoltageToSoc(double v)
        {
            // груба piecewise аппроксимація
            if (v <= 3.3) return 5;
            if (v <= 3.6) return 20;
            if (v <= 3.7) return 50;
            if (v <= 3.9) return 80;
            return 95;
        }
    }
}

ProfileManager.cs
using System;
using System.Collections.Generic;
using System.IO;
using System.Text.Json;

namespace EcoControl
{
    public class ProfileManager
    {
        private readonly string _path;
        public List<Profile> Profiles { get; private set; } = new List<Profile>();

        public ProfileManager()
        {
            var appData = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
            var dir = Path.Combine(appData, "EcoControl");
            Directory.CreateDirectory(dir);
            _path = Path.Combine(dir, "profiles.json");
            Load();
        }

        public void Load()
        {
            try
```

```
        {
            if (File.Exists(_path))
            {
                var json = File.ReadAllText(_path);
                Profiles =
JsonSerializer.Deserialize<List<Profile>>(json)
?? new List<Profile>();
            }
            else
            {
                // дефолтні профілі
                Profiles = new List<Profile>
                {
                    new Profile { Name = "Eco",
BrightnessPercent = 30, BackgroundPriority =
System.Diagnostics.ProcessPriorityClass.BelowN
ormal, CpuLimitPercent = 50, DisableWiFi =
false },
                    new Profile { Name = "Balanced",
BrightnessPercent = 60, BackgroundPriority =
System.Diagnostics.ProcessPriorityClass.Normal,
CpuLimitPercent = 75, DisableWiFi = false },
                    new Profile { Name =
"Performance", BrightnessPercent = 100,
BackgroundPriority =
System.Diagnostics.ProcessPriorityClass.Normal,
CpuLimitPercent = 100, DisableWiFi = false }
                };
                Save();
            }
        }
        catch
        {
            Profiles = new List<Profile>();
        }
    }

    public void Save()
    {
        try
        {
            var json =
JsonSerializer.Serialize(Profiles, new
JsonSerializerOptions { WriteIndented = true });
            File.WriteAllText(_path, json);
        }
        catch { }
    }
  }
}
```

Controller.cs

```
using System;
using System.Diagnostics;
using System.Linq;

namespace EcoControl
{
  // Застосування налаштувань: powercfg,
пріоритети процесів, просте очищення пам'яті
  public class Controller
  {
    public void ApplyProfile(Profile p)
    {
      try
      {
        // 1) Змінити яскравість (WMI) - деякі
системи підтримують
        SetBrightness(p.BrightnessPercent);

        // 2) Застосувати план живлення
через powercfg - тут ми не створюємо новий
GUID,
        //   замість цього можна змінити
окремі параметри або перемикатись на
precreated GUID
        //   Для прикладу викликаємо
powercfg /setacvalueindex на параметр дисплея
(приближено)
        //   Простий приклад: (треба знати
GUID підгрупи/параметра) - пропускаємо тут,
використовуємо setactive при наявності GUID.

        // 3) Знизити пріоритет фонових
процесів

ReduceBackgroundProcessPriority(p.Background
Priority);

        // 4) Якщо потрібно - відключати
WiFi (вимагатиме netsh або інструменту OEM)
        if (p.DisableWiFi)
        {
          Utils.RunCmd("netsh", "interface set
interface \"Wi-Fi\" admin=disabled");
        }
      }
      catch (Exception ex)
      {
        Debug.WriteLine("ApplyProfile error: "
+ ex.Message);
      }
    }

    private void SetBrightness(int percent)
    {
      // Спроба через WMI
(Win32_BrightnessMethods) - може не
підтримуватись на всіх пристроях
      try
      {
```

```
            var mc = new
System.Management.ManagementClass("WmiMo
nitorBrightnessMethods");
            mc.Scope = new
System.Management.ManagementScope(@"\\.\ro
ot\wmi");
            foreach
(System.Management.ManagementObject mo in
mc.GetInstances())
            {
               var inParams =
mo.GetMethodParameters("WmiSetBrightness");
               inParams["Brightness"] = percent;
               inParams["Timeout"] = 1;

mo.InvokeMethod("WmiSetBrightness",
inParams, null);
            }
         }
         catch { /* ігноруємо помилки */ }
      }

      private void
ReduceBackgroundProcessPriority(ProcessPriorit
yClass priority)
      {
         try
         {
            var current =
Process.GetCurrentProcess().Id;
            foreach (var p in
Process.GetProcesses())
            {
               try
               {
                  if (p.Id == current) continue;
                // просте правило: якщо процес
вікно не активне та використовує мало CPU,
можна знизити пріоритет
                  if (p.MainWindowHandle ==
IntPtr.Zero)
                  {
                     p.PriorityClass = priority;
                  }
               }
               catch { /* без прав у деяких
процесів */ }
            }
         }
         catch { }
      }

      // Просте "очищення пам'яті"
      public void CleanMemory()
      {
         GC.Collect();
         GC.WaitForPendingFinalizers();

         // Спроба зменшити робочі набори
інших процесів (якщо дозволено)
         foreach (var p in Process.GetProcesses())
         {
            try
            {
               if (p.Id ==
Process.GetCurrentProcess().Id) continue;
               // зменшуємо мін/макс робочий
набір (працює не завжди)
               var min = p.MinWorkingSet;
               var max = p.MaxWorkingSet;
              // встановимо мінімальний розмір
трохи нижче
               IntPtr newMin =
(IntPtr)(Environment.WorkingSet / 4);
               p.MinWorkingSet = newMin;
               p.MaxWorkingSet = newMin;
            }
            catch { }
         }
      }
   }
}
```

MainForm.cs

```
using System;
using System.Linq;
using System.Windows.Forms;

namespace EcoControl
{
   public partial class MainForm : Form
   {
      private DataCollector _collector;
      private SoCModel _soc;
      private ProfileManager _pm;
      private Controller _controller;

      public MainForm()
      {
         InitializeComponent();
         InitializeCustom();
      }

      private void InitializeCustom()
      {
         _collector = new DataCollector();
         _soc = new SoCModel();
         _pm = new ProfileManager();
         _controller = new Controller();

         _collector.TelemetryAvailable +=
OnTelemetry;
```

```
            }
_collector.Start(TimeSpan.FromSeconds(5));
        }
    }

        // Заповнимо список профілів у UI
(якщо є)
        if (_pm.Profiles.Any())
        {
            foreach (var p in _pm.Profiles)
              comboProfiles.Items.Add(p.Name);
            comboProfiles.SelectedIndex = 0;
        }

        // Подія вводу користувача
        this.MouseMove += (s, e) =>
_collector.OnUserInput();
        this.KeyPress += (s, e) =>
_collector.OnUserInput();

        // Кнопки
        btnApply.Click += (s, e) =>
ApplySelectedProfile();
        btnClean.Click += (s, e) =>
_controller.CleanMemory();
    }

    private void OnTelemetry(Telemetry t)
    {
        // оновлення SoC
        _soc.UpdateFromTelemetry(t);

        // Оновимо UI (invoke)
        this.BeginInvoke(new Action(() =>
        {
            lblBattery.Text = $"Battery:
{(t.BatteryPercent >= 0 ?
t.BatteryPercent.ToString() : "?")}%,
SoC≈{_soc.EstimatedSoC:F0}%";
            lblCpu.Text = $"CPU:
{t.CpuUsagePercent}%";
            lblUserActive.Text = t.UserActive ?
"Active" : "Idle";
        }));
    }

    private void ApplySelectedProfile()
    {
      var name = comboProfiles.SelectedItem as
string;
        var p = _pm.Profiles.FirstOrDefault(x =>
x.Name == name);
        if (p != null)
        {
            _controller.ApplyProfile(p);
          MessageBox.Show($"Profile {p.Name}
applied", "EcoControl");
        }
```