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

СЕМЕНИШИН Олександр Мирославович

Інтелектуальний модуль моніторингу продуктивності та енергоефективності моделей глибокого навчання на вбудованих обчислювальних пристроях / Intelligent Module for Monitoring the Performance and Energy Efficiency of Deep Learning Models on Embedded Computing Devices

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

Кваліфікаційна робота

Виконав студент групи КН-41
О. М. Семенишин

_____

Науковий керівник:
к.т.н., професор, В. В. Кочан

_____

Кваліфікаційну роботу
допущено до захисту:
"___"_____ 20__ р.

Завідувач кафедри
_____ **Н.В. Дзюбановська**

Тернопіль – 2026

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

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.В. Дзюбановська
«____» _____20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
СЕМЕНИШИНУ Олександру Мирославовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Інтелектуальний модуль моніторингу продуктивності та енергоефективності моделей глибокого навчання на вбудованих обчислювальних пристроях / Intelligent Module for Monitoring the Performance and Energy Efficiency of Deep Learning Models on Embedded Computing Devices
керівник роботи к.т.н., професор В.В. Кочан
затверджені наказом по університету від 01 грудня 2025 року № 894.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2026 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити
- проаналізувати особливості застосування моделей глибокого навчання на вбудованих обчислювальних пристроях;
- дослідити основні підходи до оцінювання продуктивності та енергоефективності моделей;
- виконати аналіз сучасних засобів оптимізації та моніторингу моделей на embedded- і edge-платформах;
- сформувати вимоги до інтелектуального модуля моніторингу;
- розробити структуру та алгоритми збору телеметрії, аналізу метрик і формування рекомендацій;
- реалізувати програмний прототип модуля;
- провести експериментальні дослідження роботи модуля на вбудованій платформі та оцінити його ефективність.
5. Перелік графічного матеріалу в роботі
- загальна архітектура інтелектуального модуля моніторингу;
- структурна схема збору даних та оброблення показників роботи моделі;
- схема алгоритму оцінювання стану моделі та формування рекомендацій;
- структура даних інтелектуального модуля моніторингу.

6. Консультанти розділів кваліфікаційної роботи

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

7. Дата видачі завдання 01 грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи	до 01.01. 2026 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02. 2026 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 15.03.2026 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05. 2026 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2026 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2026 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2026 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2026 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06. 2026 р.	

Студент_____ О.М. Семенишин
підпис

Керівник роботи_____ к.т.н., професор В.В. Кочан
підпис

АНОТАЦІЯ

Кваліфікаційна робота на тему «Інтелектуальний модуль моніторингу продуктивності та енергоефективності моделей глибокого навчання на вбудованих обчислювальних пристроях» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом в 79 сторінок і містить 14 ілюстрацій, 9 таблиць, 2 додатки та 38 використаних джерел.

Метою роботи є розроблення інтелектуального модуля моніторингу продуктивності та енергоефективності моделей глибокого навчання на вбудованих обчислювальних пристроях, який забезпечує збір, аналіз і оцінювання ключових характеристик роботи моделі.

Методи досліджень включають аналіз наукових джерел, системний аналіз, методи оцінювання продуктивності та енергоефективності, методи статистичного узагальнення.

Розроблено інтелектуальний модуль моніторингу продуктивності та енергоефективності моделей глибокого навчання на вбудованих обчислювальних пристроях, який забезпечує завантаження моделей, запуск їх виконання, збір системних даних, фіксацію часових показників, обчислення метрик продуктивності й енергоефективності, визначення стану моделі та формування рекомендацій щодо підвищення ефективності її роботи.

Результати дослідження можуть бути використані для побудови систем моніторингу моделей глибокого навчання на embedded- і edge-платформах, порівняння різних архітектур нейронних мереж, вибору раціонального режиму виконання моделей, розроблення програмних засобів інтелектуального аналізу продуктивності.

Ключові слова: ГЛИБОКЕ НАВЧАННЯ, ВБУДОВАНІ ОБЧИСЛЮВАЛЬНІ ПРИСТРОЇ, EDGE AI, ПРОДУКТИВНІСТЬ, ЕНЕРГОЕФЕКТИВНІСТЬ, МОНІТОРИНГ, NVIDIA JETSON, PYTORCH, TENSORRT, МОДЕЛЬ ГЛИБОКОГО НАВЧАННЯ, ПРОГРАМНИЙ МОДУЛЬ.

ANNOTATION

Qualification work on the topic «Intelligent Module for Monitoring the Performance and Energy Efficiency of Deep Learning Models on Embedded Computing Devices» for Bachelor's degree on speciality 122 «Computer Science» educational and professional program «Computer Science» is written on 79 pages and it contains 14 figures, 9 tables, 2 annexes and 38 sources.

The purpose of the work is to develop an intelligent module for monitoring the performance and energy efficiency of deep learning models on embedded computing devices, which provides the collection, analysis, and evaluation of key characteristics of model operation.

The research methods include analysis of scientific sources, system analysis, methods for evaluating performance and energy efficiency, and methods of statistical generalization.

An intelligent module for monitoring the performance and energy efficiency of deep learning models on embedded computing devices has been developed. It provides model loading, execution launch, collection of system data, recording of time-based indicators, calculation of performance and energy efficiency metrics, determination of the model state, and generation of recommendations for improving its operational efficiency.

The research results can be used to build monitoring systems for deep learning models on embedded and edge platforms, compare different neural network architectures, select a rational mode of model execution, and develop software tools for intelligent performance analysis.

Keywords: DEEP LEARNING, EMBEDDED COMPUTING DEVICES, EDGE AI, PERFORMANCE, ENERGY EFFICIENCY, MONITORING, NVIDIA JETSON, PYTORCH, TENSORRT, DEEP LEARNING MODEL, SOFTWARE MODULE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI – штучний інтелект

API – програмний інтерфейс застосування

CPU – центральний процесор

DL – глибоке навчання

Edge AI – виконання моделей штучного інтелекту на периферії мережі

EfficientNet – сімейство ефективних згорткових нейронних мереж

GPU – графічний процесор

IoT – Інтернет речей

MobileNetV2 – полегшена архітектура згорткової нейронної мережі

ONNX – відкритий формат подання моделей машинного навчання

PyTorch – програмна бібліотека для розроблення моделей глибокого навчання

ResNet18 – архітектура згорткової нейронної мережі з залишковими зв’язками

ShuffleNet V2 – полегшена архітектура згорткової нейронної мережі для мобільних і вбудованих систем

TensorRT – програмний засіб оптимізації та прискорення виконання моделей глибокого навчання

TorchVision – бібліотека моделей і засобів оброблення зображень для PyTorch

ВСТУП

Стрімке поширення моделей глибокого навчання в сучасних інформаційних системах зумовило їх активне впровадження не лише в хмарних середовищах, а й на вбудованих обчислювальних пристроях. Такі моделі застосовуються в системах комп'ютерного зору, безпілотних платформах, промисловому контролі, робототехніці, сенсорних мережах, системах Інтернету речей та інших прикладних рішеннях, де важливими є автономність, низька затримка та оброблення даних у реальному часі [3, 19, 33]. Перенесення інтелектуальної обробки безпосередньо на край мережі дає змогу зменшити залежність від хмарної інфраструктури, знизити навантаження на канали передавання даних і підвищити оперативність прийняття рішень [3, 19].

Разом із цим ефективне виконання моделей глибокого навчання на вбудованих платформах супроводжується низкою обмежень. На відміну від серверних систем, edge- та embedded-пристрої мають обмежені обчислювальні ресурси, енергетичний бюджет, пам'ять і теплові характеристики. У таких умовах навіть модель із високою точністю може виявитися непридатною для практичного використання через надмірну затримку інференсу, нестабільне використання процесорних і графічних ресурсів або підвищене енергоспоживання [1, 5, 15]. Це формує потребу не лише в оптимізації моделей, а й у постійному контролі їх фактичної поведінки на цільовій апаратній платформі.

Актуальність теми зумовлена тим, що оцінювання моделей глибокого навчання на вбудованих пристроях дедалі частіше виходить за межі традиційних метрик точності. На практиці важливими стають час інференсу, пропускна здатність, використання CPU, GPU та пам'яті, температурний режим, енергоспоживання, а також інтегральні показники енергоефективності [1, 3, 5, 12, 34]. Саме тому виникає потреба у створенні інтелектуального модуля, здатного автоматизовано збирати телеметрію, аналізувати стан виконання моделі та формувати висновки щодо її ефективності на конкретному пристрої.

Сучасні дослідження підтверджують, що для вбудованих систем вирішальне значення мають апаратно-орієнтовані підходи до оптимізації інференсу.

Використання TensorRT, ONNX, спеціалізованих бібліотек і полегшених архітектур нейронних мереж дає змогу істотно скоротити час виконання моделей і знизити обсяг ресурсів, потрібних для їх роботи [5, 16, 18, 20, 21, 32]. Значну роль відіграють також методи посттренувальної квантизації, структурного проріджування, компресії та добору компактних архітектур, зокрема MobileNet, MobileNetV2, ShuffleNet V2, SqueezeNet, EfficientNet та інших [2, 6, 8, 10, 11, 14, 22, 25, 29]. Однак навіть за наявності таких методів у багатьох прикладних рішеннях недостатньо уваги приділяється безперервному моніторингу реальної продуктивності та енергоефективності вже розгорнутої моделі.

Окремої уваги потребує екологічний та енергетичний аспект використання обчислювальних систем. Зростання складності моделей штучного інтелекту супроводжується збільшенням енергетичних витрат на їх навчання та виконання, що актуалізує дослідження у сфері energy-aware computing [5]. Для вбудованих платформ це питання має подвійне значення, оскільки енергоспоживання безпосередньо впливає не лише на вартість експлуатації, а й на автономність пристрою, теплову стабільність і надійність його роботи. Відповідно, інтелектуальний моніторинг має враховувати не один показник, а сукупність взаємопов’язаних характеристик.

Аналіз наукових праць показує, що в літературі широко висвітлено питання edge computing, реалізації глибокого навчання на платформі NVIDIA Jetson, оптимізації моделей для реального часу та benchmark-оцінювання їх роботи [1, 3, 12, 15, 23]. Також досліджуються задачі побудови інтелектуальних вузлів, розподілених сенсорних середовищ, оцінювання обчислювальної складності алгоритмів та організації edge/cloud-аналітики [13, 24, 30]. Водночас недостатньо розкритим залишається питання побудови єдиного програмного модуля, який поєднує збір системної телеметрії, аналіз параметрів інференсу та формування рекомендацій щодо підвищення продуктивності й енергоефективності моделей на вбудованому пристрої.

Отже, науково-практична проблема полягає в необхідності розроблення такого програмного засобу, який давав би змогу комплексно оцінювати роботу моделей глибокого навчання на вбудованих обчислювальних платформах з

урахуванням їх ресурсних та енергетичних обмежень. Розв’язання цієї проблеми сприятиме більш обґрунтованому вибору моделей, режимів їх виконання та засобів оптимізації в умовах edge-обчислень.

Метою роботи є розроблення інтелектуального модуля моніторингу продуктивності та енергоефективності моделей глибокого навчання на вбудованих обчислювальних пристроях, який забезпечує збір, аналіз і оцінювання ключових характеристик роботи моделі.

Для досягнення поставленої мети необхідно розв’язати такі завдання:

– проаналізувати особливості застосування моделей глибокого навчання на вбудованих обчислювальних пристроях;

– дослідити основні підходи до оцінювання продуктивності та енергоефективності моделей;

– виконати аналіз сучасних засобів оптимізації та моніторингу моделей на embedded- і edge-платформах;

– сформувати вимоги до інтелектуального модуля моніторингу;

– розробити структуру та алгоритми збору телеметрії, аналізу метрик і формування рекомендацій;

– реалізувати програмний прототип модуля;

– провести експериментальні дослідження роботи модуля на вбудованій платформі та оцінити його ефективність.

Об’єктом дослідження є процес функціонування моделей глибокого навчання на вбудованих обчислювальних пристроях.

Предметом дослідження є методи, моделі та програмні засоби моніторингу продуктивності та енергоефективності моделей глибокого навчання в умовах обмежених апаратних ресурсів.

Результати кваліфікаційної роботи апробовані на міжнародній науковій інтернет-конференції «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення», м. Тернопіль, Україна, м. Ополе, Польща, 7-8 травня 2026 р. (додаток Б).

Кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел та додатків.

1 ТЕОРЕТИЧНІ ОСНОВИ МОНІТОРИНГУ ПРОДУКТИВНОСТІ ТА ЕНЕРГОЕФЕКТИВНОСТІ МОДЕЛЕЙ ГЛИБОКОГО НАВЧАННЯ НА ВБУДОВАНИХ ПРИСТРОЯХ

1.1 Особливості застосування моделей глибокого навчання на вбудованих обчислювальних пристроях

Сучасний етап розвитку інформаційних технологій характеризується швидким переходом від централізованої обробки даних до розподілених інтелектуальних обчислень, наближених до джерела виникнення інформації. У цій парадигмі вбудовані обчислювальні пристрої та edge-платформи набувають особливого значення, оскільки дають змогу виконувати аналіз даних безпосередньо на місці їх отримання, без постійного звернення до хмарної інфраструктури [3, 19, 33]. Такий підхід є особливо важливим для систем реального часу, автономних мобільних засобів, робототехнічних платформ, сенсорних мереж, безпілотних літальних апаратів, інтелектуальних камер спостереження та промислових IoT-рішень, де затримка передавання даних у хмару може істотно знижувати ефективність функціонування системи [3, 34].

Моделі глибокого навчання сьогодні є основою великої кількості прикладних рішень у галузях комп’ютерного зору, розпізнавання образів, виявлення об’єктів, класифікації сцен, аналізу сенсорної інформації та прогнозування станів складних технічних об’єктів [7, 12]. Проте їх практичне використання на вбудованих платформах суттєво відрізняється від розгортання в серверних або хмарних середовищах. Якщо в центрах обробки даних можна компенсувати високу складність моделі значними обчислювальними ресурсами, то в умовах edge-обчислень необхідно враховувати жорсткі обмеження за продуктивністю процесора, потужністю графічного прискорювача, обсягом оперативної пам’яті, пропускною здатністю пам’яті, тепловиділенням та енергоспоживанням [1, 15, 19].

Перевагою виконання моделей глибокого навчання на вбудованому пристрої є насамперед зменшення затримки між надходженням вхідних даних і отриманням результату. У задачах виявлення подій, навігації, технічного контролю та захисту об’єктів рішення має формуватися за мілісекунди або частки секунди, тому

залежність від віддаленого сервера є небажаною [3, 23]. Крім того, локальне виконання інференсу знижує вимоги до каналів передавання даних, підвищує рівень конфіденційності й захисту інформації, а також забезпечує вищу стійкість системи до перебоїв зв'язку. Саме тому edge computing розглядається як природне середовище для розміщення інтелектуальних функцій вбудованих систем [3, 19].

Разом із тим використання моделей глибокого навчання на embedded-платформах супроводжується рядом технічних труднощів. Однією з головних проблем є невідповідність між високою обчислювальною складністю сучасних нейромереж і обмеженими можливостями апаратної платформи. Більшість класичних архітектур, що демонструють високі показники точності, зокрема VGG, ResNet, Inception чи DenseNet, початково розроблялися з орієнтацією на потужні GPU-системи [7, 9, 26, 28]. Унаслідок цього їх пряме перенесення на мікрокомп'ютери, одноплатні системи або компактні edge-пристрої часто призводить до неприйнятного часу інференсу, підвищеного навантаження на пам'ять і надмірного енергоспоживання [1, 15].

У відповідь на ці обмеження сформувався окремий напрям досліджень, пов'язаний із розробленням компактних і ресурсоефективних архітектур нейронних мереж. До таких архітектур належать MobileNet, MobileNetV2, ShuffleNet V2, SqueezeNet, EfficientNet та інші моделі, побудовані з урахуванням вимог мобільних і вбудованих платформ [8, 10, 14, 25, 29]. Їх ключовими особливостями є зменшення кількості параметрів, скорочення числа обчислювальних операцій, використання глибинно-роздільних згорток, вузьких пляшкових блоків, оптимізованих механізмів масштабування та практичних правил проєктування компактних CNN. Саме ці властивості дають змогу досягати прийнятного компромісу між точністю та швидкодією на edge-пристроях.

Для вбудованих систем важливою є не лише абсолютна точність моделі, а збалансованість кількох характеристик одночасно. На практиці модель повинна забезпечувати достатній рівень якості прогнозування за обмеженого часу інференсу, допустимого обсягу пам'яті та контрольованого споживання енергії. Таким чином, ефективність моделі в умовах embedded-обчислень визначається не одним критерієм, а сукупністю взаємопов'язаних показників. Серед них

найчастіше аналізуються latency, throughput, кількість параметрів, FLOPS, завантаження CPU/GPU, використання RAM, температура пристрою та енерговитрати на один запуск моделі [1, 5, 15, 34]. Саме багатокритеріальний характер оцінювання робить задачу вибору моделі для вбудованої платформи значно складнішою, ніж у звичайних серверних сценаріях.

Окремої уваги потребує енергетичний аспект. Для автономних вбудованих систем, що живляться від акумуляторів або працюють у режимі тривалого безперервного функціонування, енергоспоживання є критично важливим параметром [5, 19]. Навіть якщо модель демонструє прийнятний час інференсу, її практичне використання може бути недоцільним через швидке розрядження живлення, перегрівання або нестабільність продуктивності під тривалим навантаженням. Тому сучасні підходи до edge AI дедалі частіше орієнтуються на energy-aware deployment, у межах якого оцінюється не лише швидкість, а й енергоефективність виконання моделі [1, 5, 34]. Для цього використовують інтегральні показники, наприклад кількість кадрів за секунду на ват, енерговитрати на один інференс або співвідношення між точністю та ресурсною вартістю.

Ще однією особливістю застосування моделей глибокого навчання на вбудованих пристроях є висока залежність результатів від конкретної апаратно-програмної платформи. Одна й та сама модель може демонструвати різну швидкодію та різне енергоспоживання залежно від архітектури GPU, версії драйверів, бібліотек прискорення, формату подання ваг, пакетного розміру та механізму оптимізації графа обчислень [12, 15, 16, 18, 20, 21]. З цієї причини результати, отримані в абстрактному програмному середовищі, не завжди відтворюються на цільовому пристрої. Практична придатність моделі має визначатися безпосередньо в тих умовах, у яких вона буде реально експлуатуватися.

У цьому контексті особливої популярності набули платформи сімейства NVIDIA Jetson, які поєднують компактність, наявність GPU-прискорення та підтримку спеціалізованих засобів оптимізації інференсу [15, 16, 18]. Застосування TensorRT, ONNX та інтегрованих інструментів NVIDIA дає змогу зменшити накладні витрати виконання, об’єднувати операції, використовувати нижчу

точність обчислень і краще адаптувати модель до властивостей цільового прискорювача [12, 18, 20, 32]. Для вбудованих пристроїв це має принципове значення, оскільки навіть незначне зниження затримки або енерговитрат у кожному циклі інференсу в сукупності дає суттєвий виграш у довготривалій експлуатації системи.

На продуктивність моделей у вбудованому середовищі значно впливають методи оптимізації. Найпоширенішими серед них є квантизація, проріджування, компресія параметрів, оптимізація структури графа та використання полегшених архітектур [2, 6, 11, 22, 31]. Квантизація дає змогу зменшити розрядність ваг і активацій, що знижує споживання пам'яті та пришвидшує обчислення. Проріджування та стиснення моделі зменшують кількість надлишкових параметрів. Оптимізація графа обчислень скорочує накладні витрати на виконання окремих операцій. Разом ці методи утворюють практичний інструментарій, необхідний для адаптації глибоких мереж до embedded-середовища.

Проте навіть оптимізована модель не гарантує стабільної й ефективної роботи без належного моніторингу. Реальні умови функціонування вбудованого пристрою можуть змінюватися в часі: змінюється температура, накопичується навантаження, виникають фонові процеси, модифікується режим живлення або частота процесора. За таких умов показники інференсу можуть відхилятися від лабораторних значень. Саме тому для практичного застосування моделей глибокого навчання на вбудованих платформах необхідні засоби безперервного спостереження за станом системи, які дають змогу фіксувати ключові метрики, виявляти деградацію продуктивності та формувати рекомендації щодо коригування режиму роботи [13, 17, 24, 30].

Слід також враховувати, що вбудовані інтелектуальні системи часто є частиною ширшої інфраструктури Інтернету речей. У такому середовищі пристрій не лише виконує модель, а й взаємодіє з сенсорами, мережевими модулями, керуючими елементами та зовнішніми сервісами [19, 33]. Це означає, що продуктивність глибокого навчання необхідно розглядати не ізольовано, а як компонент загальної системної поведінки. Неефективний інференс може впливати на затримку реагування, інтенсивність обміну даними, частоту опитування

сенсорів і загальний енергетичний баланс системи. Отже, задача оцінювання моделей на вбудованих пристроях природно переходить у задачу інтелектуального моніторингу комплексного функціонування пристрою.

Таким чином, застосування моделей глибокого навчання на вбудованих обчислювальних пристроях має низку характерних особливостей: обмеженість ресурсів, багатокритеріальність оцінювання ефективності, критичність затримок, залежність від конкретної апаратної платформи, необхідність використання компактних архітектур і методів оптимізації, а також потребу в постійному моніторингу продуктивності та енергоспоживання. Саме сукупність цих факторів формує підґрунтя для розроблення спеціалізованого інтелектуального модуля, здатного аналізувати роботу моделей глибокого навчання в реальних умовах експлуатації на embedded-пристрої та підтримувати прийняття рішень щодо підвищення ефективності їх функціонування.

Отже, застосування моделей глибокого навчання на вбудованих обчислювальних пристроях є перспективним напрямом розвитку edge AI, однак супроводжується значними ресурсними та енергетичними обмеженнями. Ефективність таких моделей визначається не лише точністю, а й часом інференсу, використанням апаратних ресурсів, тепловими характеристиками та енерговитратами.

1.2 Підходи до оцінювання продуктивності та енергоефективності моделей глибокого навчання

Оцінювання моделей глибокого навчання для вбудованих обчислювальних пристроїв істотно відрізняється від традиційного підходу, який тривалий час домінував у задачах машинного навчання. Якщо в класичних дослідженнях основна увага зосереджувалася переважно на точності класифікації, повноті, точності виявлення або інших показниках якості прогнозу, то для embedded- і edge-платформ цього вже недостатньо. У реальних умовах експлуатації модель має не лише формувати коректний результат, а й виконуватися в межах доступного енергетичного бюджету, пам’яті та обчислювальної потужності, забезпечуючи при

цьому стабільну роботу системи в реальному часі [1, 3, 15].

У зв’язку з цим оцінювання ефективності моделей на вбудованих пристроях набуває багатокритеріального характеру. До уваги беруться не тільки метрики якості прогнозування, а й час інференсу, пропускна здатність, навантаження на CPU і GPU, використання оперативної та відеопам’яті, температура, споживана потужність, повна енергія на один цикл оброблення, а також стійкість показників у тривалому режимі роботи [5, 12, 34]. Таким чином, оцінювання переходить від вузького аналізу «наскільки точно працює модель» до ширшого питання «наскільки придатна ця модель для конкретної апаратної платформи та сценарію застосування».

Першу групу становлять показники функціональної якості моделі. До них належать accuracy, precision, recall, F1-score, mAP та інші метрики, які характеризують правильність результату. Для вбудованих систем ці показники залишаються важливими, оскільки оптимізація моделі не повинна критично погіршувати якість розпізнавання або прогнозування [7, 25, 28]. Проте на практиці висока точність сама по собі ще не гарантує придатності моделі до використання. Наприклад, архітектура може демонструвати високі результати на тестовому наборі даних, але виявитися надто повільною для реального часу або занадто затратною за енергією при запуску на вбудованому модулі. Тому функціональні показники доцільно розглядати як необхідну, але не достатню умову ефективності.

Друга група охоплює показники швидкодії та обчислювальної продуктивності. Найуживанішими серед них є час інференсу для одного зразка, середня затримка оброблення, кількість кадрів за секунду, пропускна здатність за певного розміру пакета та стабільність часу відповіді [1, 23, 32]. У задачах комп’ютерного зору, відеоаналітики або управління автономними об’єктами саме latency часто є визначальним параметром, оскільки навіть невелике зростання затримки може зробити систему непридатною до практичного використання. Для таких сценаріїв ефективною вважається не просто швидка модель, а модель із передбачуваною швидкодією без різких коливань часу виконання під навантаженням.

Окреме місце посідають структурні характеристики моделі, що дають змогу

попередньо оцінити її ресурсомісткість. До них належать кількість параметрів, число арифметичних операцій, FLOPS, глибина мережі, розмір вхідного тензора, а також типи обчислювальних блоків. Саме ці характеристики часто використовують як первинний орієнтир під час порівняння архітектур, особливо якщо йдеться про MobileNet, ShuffleNet, SqueezeNet, EfficientNet, ResNet, VGG чи Inception [7, 8, 10, 14, 26, 28, 29]. Водночас прямий зв'язок між кількістю параметрів і реальною швидкодією не завжди є лінійним. Менша модель не завжди працює швидше, якщо її структура погано узгоджується з особливостями конкретного прискорювача. Тому структурні показники є корисними, але не можуть замінити реального вимірювання продуктивності на цільовому пристрої.

Третя група включає метрики використання апаратних ресурсів. Для вбудованих платформ важливо відстежувати завантаження центрального процесора, графічного прискорювача, обсяг використаної оперативної пам'яті, навантаження на підсистему зберігання даних і шини обміну. Такий аналіз дає змогу виявити вузькі місця системи, коли низька продуктивність зумовлена не стільки самою моделлю, скільки нераціональною організацією оброблення даних, накладними витратами фреймворку або недостатньою пропускною здатністю пам'яті [3, 15, 18]. Для задач моніторингу це особливо важливо, оскільки один і той самий показник затримки може мати різну природу: перевантаження GPU, нестачу RAM, надмірне копіювання тензорів або теплове тротлінг-обмеження.

Четверта група пов'язана з енергоефективністю виконання моделі. У межах цього підходу оцінюється не тільки споживана потужність у певний момент часу, а й сумарна енергія, витрачена на один інференс, один кадр або один оброблений пакет даних [5, 27, 34]. На відміну від серверних систем, для яких зростання енергоспоживання часто компенсується масштабом інфраструктури, у вбудованих пристроях воно безпосередньо впливає на автономність, нагрівання та стабільність роботи. Тому важливо розрізняти два близькі, але не тотожні показники: потужність як миттєве споживання і енергію як інтегральну витрату за час виконання. Модель може споживати вищу потужність, але завершувати інференс настільки швидко, що загальні енерговитрати будуть нижчими, ніж у повільнішої альтернативи. Саме тому коректне оцінювання енергоефективності має

ґрунтуватися на поєднанні часових і енергетичних показників.

На практиці для інтегрального порівняння моделей часто використовують похідні показники ефективності. До них належать кадри за секунду на ват, енергія на один інференс, співвідношення між точністю та енергоспоживанням, а також умовні індекси, що поєднують декілька нормалізованих метрик в один підсумковий бал [1, 5]. Такі показники є корисними тоді, коли потрібно не просто описати поведінку системи, а вибрати найкращу конфігурацію серед кількох альтернатив. Наприклад, одна модель може бути точнішою, але менш енергоефективною, а інша – дещо слабшою за точністю, проте значно вигіднішою в автономному режимі. У подібних ситуаціях потрібне саме багатокритеріальне оцінювання, а не однофакторне порівняння.

Важливим підходом до оцінювання є benchmark-тестування. Його сутність полягає у виконанні стандартизованого набору експериментів за однакових умов для кількох моделей, конфігурацій або платформ. Такий підхід дає змогу отримати порівнювані результати та виявити закономірності, які неочевидні під час одиничних запусків [1, 23]. У цій сфері особливу роль відіграють MLPerf Inference і MLPerf Tiny, які задають орієнтири для оцінювання продуктивності ML-систем у різних класах пристроїв [1, 23]. Хоча кваліфікаційна робота не потребує повного відтворення цих стандартів, сама логіка benchmark-підходу є доцільною: фіксований набір моделей, однакові вхідні дані, однакові режими запуску, багаторазові вимірювання й усереднення результатів.

Ще один важливий підхід пов’язаний з оцінюванням впливу оптимізацій. Для вбудованих систем практичний інтерес становить не лише аналіз базової моделі, а й порівняння її роботи до і після застосування TensorRT, ONNX-конвертації, квантизації, посттренувальної оптимізації або проріджування [2, 6, 11, 12, 18, 20, 31, 32]. У цьому випадку оцінювання має показати, якою мірою оптимізація змінює затримку, пропускну здатність, пам’ять і споживання енергії, а також чи не виникає критичної втрати якості. Саме такий підхід є найбільш прикладним для побудови інтелектуального модуля моніторингу, оскільки дає змогу не тільки спостерігати за станом моделі, а й формувати рекомендації щодо більш доцільного режиму її виконання.

Суттєве значення має і динамічне оцінювання в часі. Разові вимірювання не завжди відображають реальну поведінку системи, оскільки у вбудованому середовищі продуктивність може змінюватися через нагрівання, коливання частот, фонові процеси або зміну режиму живлення. Тому коректне дослідження має включати серію вимірювань, аналіз середніх значень, дисперсії, пікових навантажень та можливої деградації характеристик у тривалому режимі [17, 24]. Для задач моніторингу це означає, що система повинна не просто фіксувати окремі значення, а накопичувати часові ряди метрик і аналізувати їх тенденції.

На рівні інструментальної реалізації оцінювання продуктивності та енергоефективності спирається на можливості конкретної платформи. Для сімейства NVIDIA Jetson вагоме значення мають засоби телеметрії, зокрема системні утиліти контролю завантаження та температури, документація щодо TensorRT і засоби роботи з оптимізованими моделями [16–18, 20, 21]. Саме наявність таких засобів робить можливим створення модуля, який поєднує збір низькорівневих системних показників із вимірюванням параметрів інференсу нейромережі. У вбудованих IoT-системах подібний підхід логічно продовжує ідеї інтелектуальних вузлів, сенсорних мереж і edge/cloud-аналітики, де оцінювання стану системи є невід'ємною частиною її функціонування [13, 24, 30, 33].

Отже, сучасні підходи до оцінювання моделей глибокого навчання на вбудованих пристроях можна звести до кількох взаємодоповнювальних напрямів: оцінювання функціональної якості, вимірювання швидкодії, аналізу ресурсомісткості, дослідження енергоефективності, benchmark-порівняння, вивчення впливу оптимізацій і динамічного моніторингу в часі. Найбільш обґрунтованим для практичних embedded-систем є саме комплексний підхід, у межах якого модель оцінюється одночасно за кількома групами показників.

Таким чином, оцінювання продуктивності та енергоефективності моделей глибокого навчання на вбудованих обчислювальних пристроях має здійснюватися комплексно, з урахуванням якості прогнозування, швидкодії, використання апаратних ресурсів і енергетичних витрат. Для практичних систем найбільш доцільним є benchmark-підхід із багаторазовими вимірюваннями та аналізом впливу оптимізацій.

1.3 Аналіз сучасних засобів оптимізації та моніторингу моделей на edge- і embedded-платформах

Ефективне використання моделей глибокого навчання на вбудованих обчислювальних пристроях значною мірою залежить не лише від вибору архітектури нейронної мережі, а й від застосування спеціалізованих засобів оптимізації та моніторингу. У середовищі edge- і embedded-обчислень саме програмно-апаратна адаптація моделі до цільової платформи визначає, чи буде досягнуто необхідної швидкодії, стабільності та прийнятного рівня енергоспоживання [3, 12, 15]. Тому сучасні інструменти в цій сфері доцільно розглядати у двох взаємопов'язаних напрямах: засоби оптимізації виконання моделі та засоби контролю її фактичної поведінки під час роботи на пристрої.

Одним із базових середовищ розроблення та підготовки моделей залишається PyTorch, який широко використовується для навчання, тестування та первинного інференсу моделей глибокого навчання. Для практики edge AI важливо, що екосистема PyTorch містить готові реалізації багатьох популярних архітектур, зокрема ResNet, MobileNet, ShuffleNet, EfficientNet та інших, що спрощує їх подальше порівняння і перенесення на embedded-платформи [7, 8, 14, 21, 29]. Водночас сам PyTorch у базовому вигляді не завжди забезпечує максимальну швидкодію на вбудованому пристрої, тому в реальних системах він часто використовується як вихідне середовище, після чого модель конвертується у формат, придатний для подальшої низькорівневої оптимізації [12, 20, 21].

Ключову роль у забезпеченні сумісності між різними фреймворками та інструментами відіграє ONNX. Цей формат дає змогу експортувати модель із середовища навчання в уніфіковане подання, зручне для подальшого розгортання на різних програмно-апаратних платформах [20]. Для edge- і embedded-сценаріїв це має практичне значення, оскільки ONNX виступає проміжною ланкою між етапом створення моделі та етапом апаратно-орієнтованої оптимізації. Такий підхід знижує залежність від одного фреймворку, спрощує відтворюваність експериментів і робить процедуру розгортання більш керованою [12, 20].

Найбільш поширеним спеціалізованим засобом оптимізації інференсу для

платформ NVIDIA є TensorRT. Його призначення полягає в тому, щоб адаптувати модель до властивостей конкретного GPU-прискорювача шляхом оптимізації графа обчислень, злиття операцій, вибору ефективних ядер, підтримки зниженої точності та скорочення накладних витрат під час виконання [12, 18]. Праці, присвячені Jetson-платформам, показують, що використання TensorRT є одним із найрезультативніших підходів для зменшення часу інференсу на embedded-пристроях [12, 15, 32]. Водночас ефективність цього інструмента залежить від типу моделі, формату даних, версії програмного стеку та особливостей цільового пристрою, тому оптимізація не може зводитися до автоматичного перетворення моделі без подальшого експериментального контролю [1, 27].

До сучасних засобів оптимізації належать також методи квантизації, проріджування та компресії. Їх застосування дозволяє зменшити обсяг моделі, прискорити виконання та знизити вимоги до пам’яті. Посттренувальна квантизація дає можливість виконувати обчислення зі зниженою розрядністю без повного перенавчання моделі, а методи integer-only inference роблять можливим ефективне використання цілочисельних операцій на цільовому апаратному забезпеченні [2, 11, 31]. Поряд із цим підходи до стискання мереж і видалення надлишкових параметрів, зокрема deep compression та pruning, забезпечують додаткове скорочення обчислювальної складності [6, 22]. Для embedded-середовища це особливо важливо, оскільки виграш досягається не лише за рахунок меншого часу інференсу, а й через зниження навантаження на пам’ять та зменшення енергетичних витрат.

Однак оптимізація моделі не є універсальною та безумовно корисною за будь-яких обставин. Частина методів може призводити до зниження точності, зміни характеру помилок або нестабільної поведінки моделі на реальних даних. Саме тому сучасні засоби оптимізації слід розглядати не як самоціль, а як інструменти пошуку прийнятного компромісу між точністю, швидкодією та енергоефективністю [4, 11, 31]. У цьому контексті особливе значення мають benchmark-дослідження, у яких одна й та сама модель порівнюється до і після оптимізації в однакових умовах запуску [1, 23]. Такий підхід дозволяє встановити, чи є застосована оптимізація дійсно доцільною для конкретного класу задач і

конкретної embedded-платформи.

Поряд із засобами оптимізації не менш важливими є засоби моніторингу. На платформі NVIDIA Jetson базовий рівень спостереження за станом системи забезпечується через документацію та вбудовані утиліти розробника, зокрема засоби контролю параметрів пристрою, температури, частот, використання процесорних і графічних ресурсів [16–18]. Для задач кваліфікаційної роботи особливо значущою є утиліта Tegrastats, яка дозволяє отримувати дані про завантаження CPU і GPU, використання пам’яті, температурні показники та інші параметри системного стану [17]. Її перевага полягає в простоті інтеграції у власний програмний модуль моніторингу та можливості періодичного збору телеметрії без суттєвого втручання в процес виконання моделі.

Проте низькорівневі системні утиліти не забезпечують повної картини ефективності моделі. Вони показують стан апаратних ресурсів, але не дають безпосередньо оцінки часу інференсу, продуктивності на рівні кадрів за секунду чи енергії на один запуск. Тому на практиці сучасні системи моніторингу формуються як комбінація двох рівнів: системного рівня, де збирається телеметрія пристрою, та модельного рівня, де вимірюються параметри інференсу, затримка, частота оброблення та якість результату [1, 12, 17]. Саме поєднання цих двох груп показників створює основу для коректного оцінювання продуктивності й енергоефективності моделі на embedded-пристрої.

У наукових працях, присвячених edge AI, окремо підкреслюється значення інтелектуального аналізу телеметрії. Йдеться не лише про фіксацію числових значень, а про виявлення закономірностей, перевантажень, деградації продуктивності, наближення до теплових обмежень і неефективних режимів виконання [13, 24, 30]. Ця логіка добре узгоджується з підходами, які історично використовувалися в інтелектуальних сенсорних мережах і розподілених системах спостереження, де система не просто накопичує дані, а інтерпретує їхній стан і підтримує прийняття рішень [13, 24, 30, 33]. Для теми даної роботи це означає, що модуль моніторингу повинен не обмежуватися роллю реєстратора метрик, а виконувати функцію аналітичного ядра.

Значущим напрямом є також стандартизоване тестування та порівняння. Для

цього використовуються підходи на основі benchmark-наборів, зокрема MLPerf Inference та MLPerf Tiny, які задають загальні принципи оцінювання продуктивності моделей у різних класах обчислювальних систем [1, 23]. Такі засоби є корисними не лише як готові еталони, а і як методична основа для побудови власної схеми експериментів. У межах кваліфікаційної роботи це означає доцільність фіксованого набору моделей, єдиного середовища запуску, багаторазового повторення вимірювань і подальшого узагальнення результатів у вигляді порівняльних таблиць та графіків [1, 23, 34].

Слід зазначити, що сучасні засоби оптимізації та моніторингу розвиваються нерівномірно. Інструменти оптимізації, такі як TensorRT, ONNX-конвеєри або квантизація, вже достатньо добре інтегровані в екосистему edge AI. Натомість інструменти моніторингу часто залишаються фрагментарними: окремо існують засоби вимірювання часу, окремо системна телеметрія, окремо журнали виконання, але відсутній універсальний модуль, який поєднував би ці компоненти в єдину систему оцінювання [3, 15, 35]. Україномовні дослідження також здебільшого зосереджені або на порівнянні програмно-апаратного забезпечення, або на аналізі продуктивності окремих класів моделей для вбудованих систем, але не на побудові цілісного інтелектуального модуля моніторингу [34, 35].

Таким чином, аналіз сучасних засобів показує, що для edge- і embedded-платформ уже існує достатня база інструментів для оптимізації моделей: PyTorch як середовище підготовки, ONNX як формат перенесення, TensorRT як засіб прискорення інференсу, а також підходи квантизації, стискання та проріджування. Поряд із цим доступні базові засоби системного моніторингу на рівні апаратної платформи. Проте залишається актуальною проблема інтеграції цих можливостей у єдине програмне рішення, здатне автоматично збирати телеметрію, оцінювати продуктивність моделі, аналізувати енергоефективність і формувати аналітичні рекомендації щодо оптимального режиму її використання.

1.4 Постановка задачі розроблення інтелектуального модуля моніторингу

Аналіз предметної області показав, що застосування моделей глибокого навчання на вбудованих обчислювальних пристроях пов’язане з необхідністю одночасного врахування кількох груп показників: якості роботи моделі, швидкодії інференсу, використання апаратних ресурсів, температурного стану та енергоспоживання [1, 3, 5, 15]. Існуючі засоби оптимізації та моніторингу здебільшого розв’язують окремі задачі, але не забезпечують цілісного аналізу ефективності моделі в реальних умовах роботи embedded-платформи [12, 17, 18].

У зв’язку з цим виникає задача розроблення інтелектуального модуля моніторингу, призначеного для автоматизованого збору, збереження, оброблення та аналізу показників функціонування моделей глибокого навчання на вбудованому пристрої. Такий модуль повинен поєднувати дані про час інференсу, завантаження CPU і GPU, використання пам’яті, температурні параметри, а також показники енергоспоживання в межах єдиного програмного контуру [17, 21, 27].

Основними завданнями модуля є:

– збір телеметрії з вбудованої платформи під час виконання моделі;

– вимірювання ключових показників продуктивності;

– оцінювання енергоефективності роботи моделі;

– виявлення неефективних режимів функціонування;

– формування рекомендацій щодо оптимізації запуску моделі або вибору більш доцільної конфігурації.

Отже, у межах кваліфікаційної роботи необхідно спроєктувати структуру інтелектуального модуля моніторингу, обґрунтувати набір контрольованих показників, розробити алгоритм їх аналізу та реалізувати програмний прототип для експериментального дослідження на вбудованій обчислювальній платформі. Це створить основу для підвищення ефективності використання моделей глибокого навчання в системах edge AI та Інтернету речей [19, 33, 35].

Тому, метою роботи є розроблення інтелектуального модуля моніторингу продуктивності та енергоефективності моделей глибокого навчання на вбудованих обчислювальних пристроях, який забезпечує збір, аналіз і оцінювання ключових

характеристик роботи моделі під час виконання моделі.

Для досягнення поставленої мети необхідно розв’язати такі завдання:

– проаналізувати особливості застосування моделей глибокого навчання на вбудованих обчислювальних пристроях;

– дослідити основні підходи до оцінювання продуктивності та енергоефективності моделей;

– виконати аналіз сучасних засобів оптимізації та моніторингу моделей на embedded- і edge-платформах;

– сформувати вимоги до інтелектуального модуля моніторингу;

– розробити структуру та алгоритми збору телеметрії, аналізу метрик і формування рекомендацій;

– реалізувати програмний прототип модуля;

– провести експериментальні дослідження роботи модуля на вбудованій платформі та оцінити його ефективність.

Об’єктом дослідження є процес функціонування моделей глибокого навчання на вбудованих обчислювальних пристроях.

Предметом дослідження є методи, моделі та програмні засоби моніторингу продуктивності та енергоефективності моделей глибокого навчання в умовах обмежених апаратних ресурсів.

2 ПРОЄКТУВАННЯ ІНТЕЛЕКТУАЛЬНОГО МОДУЛЯ МОНІТОРИНГУ ПРОДУКТИВНОСТІ ТА ЕНЕРГОЕФЕКТИВНОСТІ

2.1 Формування вимог до модуля моніторингу

Розроблення інтелектуального модуля моніторингу продуктивності та енергоефективності моделей глибокого навчання на вбудованих обчислювальних пристроях потребує попереднього формування чітких вимог до його функцій, структури та режимів роботи. Такі вимоги мають враховувати особливості предметної області, обмежені ресурси embedded-платформи, необхідність безперервного збору телеметрії, а також потребу в одержанні узагальненої оцінки стану моделі під час її виконання [3, 5, 15]. На відміну від звичайних засобів профілювання, запропонований модуль повинен не лише фіксувати окремі показники, а й забезпечувати їх системний аналіз та підтримку прийняття рішень щодо підвищення ефективності роботи моделі [13, 24, 30].

Насамперед доцільно визначити призначення модуля. Його основною метою є автоматизований контроль роботи моделі глибокого навчання на вбудованому пристрої з одночасним оцінюванням швидкодії, навантаження на апаратні ресурси та енергетичних характеристик. Отже, модуль має функціонувати як проміжна інтелектуальна ланка між моделлю та апаратно-програмним середовищем, у якому ця модель використовується. Такий підхід дає змогу перейти від простого запуску нейромережі до керованого виконання, коли користувач або розробник отримує не лише результат роботи моделі, а й інформацію про ефективність обраного режиму її функціонування [1, 12, 17].

Відповідно до поставленої мети можна сформувати основні вимоги до модуля (таблиця 2.1).

З урахуванням визначених вимог розроблено загальну архітектуру системи (рисунок 2.1). Її раціонально реалізувати як багатокомпонентну структуру, у межах якої кожен блок виконує окрему функцію. Першим компонентом є модуль запуску моделі, який відповідає за ініціалізацію нейромережі, подання вхідних даних і фіксацію часових параметрів виконання. Другим компонентом є модуль збору системної телеметрії, який у фоновому режимі одержує дані про стан пристрою з

використанням системних засобів моніторингу.

Таблиця 2.1 – Основні вимоги до модуля моніторингу

Група вимог	Зміст вимоги	Призначення
Функціональні	Збір телеметрії з вбудованого пристрою під час виконання моделі	Отримання даних про стан системи в реальному часі
Функціональні	Вимірювання часу виконання моделі	Оцінювання швидкодії та затримки оброблення
Функціональні	Визначення завантаження CPU, GPU та пам’яті	Аналіз використання апаратних ресурсів
Функціональні	Контроль температурних показників і енергоспоживання	Оцінювання теплового стану та енергоефективності
Функціональні	Збереження результатів моніторингу у структурованому вигляді	Подальший аналіз, порівняння та візуалізація
Аналітичні	Виявлення неефективних режимів роботи моделі	Визначення перевантаження, нестабільності та надмірних витрат енергії
Аналітичні	Формування рекомендацій щодо оптимізації	Підтримка прийняття рішень щодо вибору моделі або режиму її роботи
Нефункціональні	Низькі накладні витрати самого модуля	Мінімізація впливу моніторингу на результати вимірювання
Нефункціональні	Модульність та розширюваність архітектури	Можливість додавання нових метрик, моделей і способів аналізу
Нефункціональні	Сумісність із вбудованими платформами типу NVIDIA Jetson	Практична придатність до реального використання
Нефункціональні	Зрозуміле подання результатів	Полегшення інтерпретації отриманих показників

Третім компонентом є модуль оброблення та збереження метрик, призначений для синхронізації часових рядів, обчислення середніх, пікових і похідних показників. Четвертим компонентом виступає аналітичний модуль, який формує підсумкову оцінку продуктивності та енергоефективності, а також рекомендації для користувача. П’ятим компонентом є інтерфейс представлення результатів, через який відображаються таблиці, журнали, графіки та узагальнені висновки.

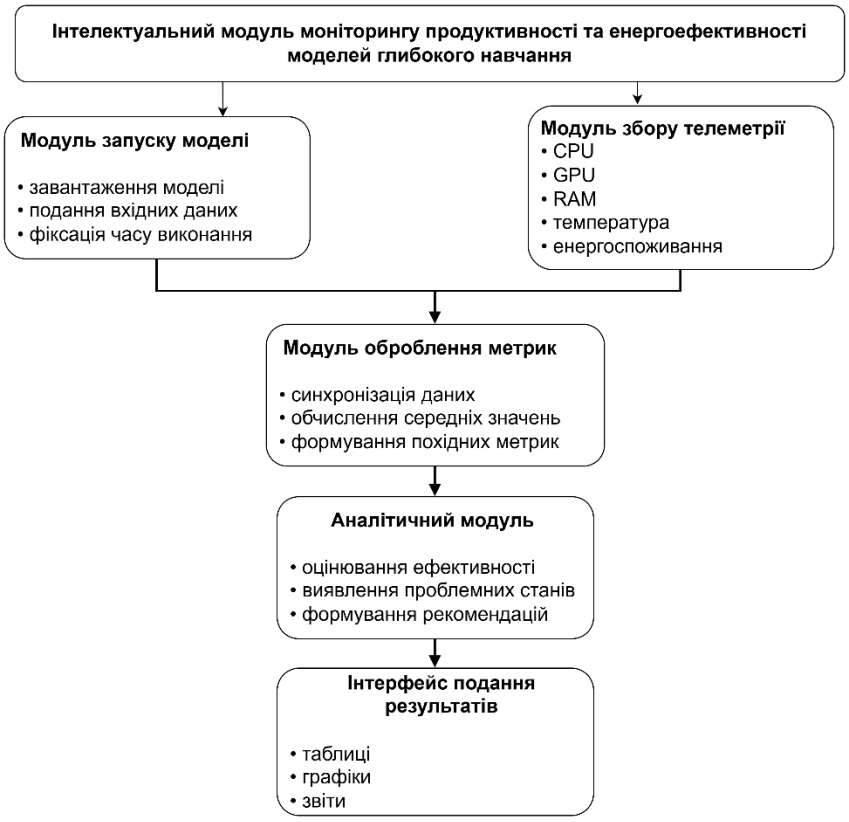

Рисунок 2.1 – Загальна архітектура інтелектуального модуля моніторингу

Логіка взаємодії цих компонентів є такою. Спочатку користувач обирає модель і режим її запуску. Після цього модуль запуску ініціює виконання моделі на вбудованому пристрої. Паралельно модуль збору телеметрії починає фіксувати системні параметри через певний інтервал часу. Після завершення серії запусків усі зібрані дані передаються до модуля оброблення, де перетворюються у впорядкований набір показників. Далі аналітичний модуль оцінює ці показники за визначеними правилами та формує висновок щодо ефективності роботи моделі. Підсумковий результат подається користувачеві у зручному вигляді для подальшого аналізу. Така організація забезпечує послідовність і зрозумілість функціонування системи.

З позиції інформаційного наповнення доцільно виділити кілька груп контрольованих показників. До першої групи належать показники швидкодії: середній час виконання моделі, мінімальний і максимальний час, кількість оброблених кадрів за секунду. До другої групи належать показники ресурсного стану: завантаження CPU, завантаження GPU, використання RAM. До третьої групи входять теплові й енергетичні характеристики: температура процесорних

вузлів, температура графічного прискорювача, споживана потужність або похідні показники енергоефективності. До четвертої групи можна віднести інтегральні характеристики, наприклад співвідношення між продуктивністю та енергоспоживанням або зведений індекс ефективності. Така структура показників дозволяє оцінювати модель комплексно, а не за одним окремим критерієм [1, 5, 17, 34].

Під час проєктування архітектури слід урахувати і режими використання модуля. Перший режим – експериментальний, коли модуль застосовується для порівняння кількох моделей або варіантів оптимізації. Другий режим – діагностичний, коли аналізується поточний стан уже розгорнутої моделі з метою виявлення вузьких місць. Третій режим – рекомендаційний, коли на основі накопичених показників модуль пропонує кращу конфігурацію запуску. Саме поєднання цих режимів робить розроблюваний модуль не лише засобом вимірювання, а й засобом інтелектуальної підтримки експлуатації моделей глибокого навчання на вбудованих пристроях.

Отже, формування вимог до модуля моніторингу показують, що розроблюване рішення повинно бути комплексним, модульним і адаптованим до ресурсних обмежень embedded-платформи. Воно має забезпечувати збір даних, вимірювання параметрів роботи моделі, оброблення та збереження метрик, аналітичне оцінювання та представлення результатів у зручному вигляді.

2.2 Розроблення схеми збору даних та оброблення показників роботи моделі

У межах проєктування інтелектуального модуля моніторингу розроблено структуру збору даних та оброблення показників роботи моделі, яка забезпечила одержання узгодженої інформації про стан вбудованого пристрою та параметри виконання моделі глибокого навчання. Побудована структура дала змогу поєднати системні характеристики платформи з показниками швидкодії моделі в межах єдиного потоку даних, придатного для подальшого аналітичного оцінювання [1, 12, 17].

У структурі збору даних виділено дві основні групи показників. До першої

групи віднесено системні дані, які охоплюють завантаження CPU, завантаження GPU, використання оперативної пам’яті, температурні показники та параметри енергоспоживання вбудованого пристрою. До другої групи віднесено показники роботи моделі, зокрема час виконання одного запуску, середній час оброблення серії вхідних даних і продуктивність. Такий поділ дав змогу розмежувати характеристики стану апаратної платформи та безпосередні результати виконання моделі, а також забезпечив їх подальше спільне аналізування [16–18, 34].

Збір даних організовано за принципом періодичного зчитування показників під час виконання моделі. У процесі роботи системи модуль моніторингу фіксує системні параметри пристрою через визначені часові інтервали, а також реєструє часові характеристики запуску моделі. Для кожного запису сформовано часову мітку, що забезпечило синхронізацію між змінами стану обчислювальної платформи та відповідними етапами виконання моделі. Такий підхід усунув розрізненість показників і сформував основу для побудови впорядкованих часових рядів.

Після завершення етапу збору виконано первинне оброблення даних. На цьому етапі забезпечено очищення неповних або некоректних значень, узгодження формату показників, агрегування даних та обчислення основних статистичних характеристик. Для кожної групи даних визначено середні, мінімальні та максимальні значення, що дало змогу перейти від набору окремих вимірювань до узагальненого представлення стану системи в межах одного циклу моніторингу.

У межах підсистеми оброблення також сформовано похідні метрики, які характеризують ефективність роботи моделі більш повно. До таких метрик віднесено середню продуктивність, зміну температури впродовж виконання, співвідношення між досягнутою продуктивністю та енергоспоживанням, а також інтегральні показники ефективності. Сформовані похідні характеристики підвищили інформативність результатів моніторингу та створили основу для виявлення неефективних режимів роботи моделі [5, 17, 27].

Структуру збору даних та оброблення показників роботи моделі наведено на рисунку 2.2.

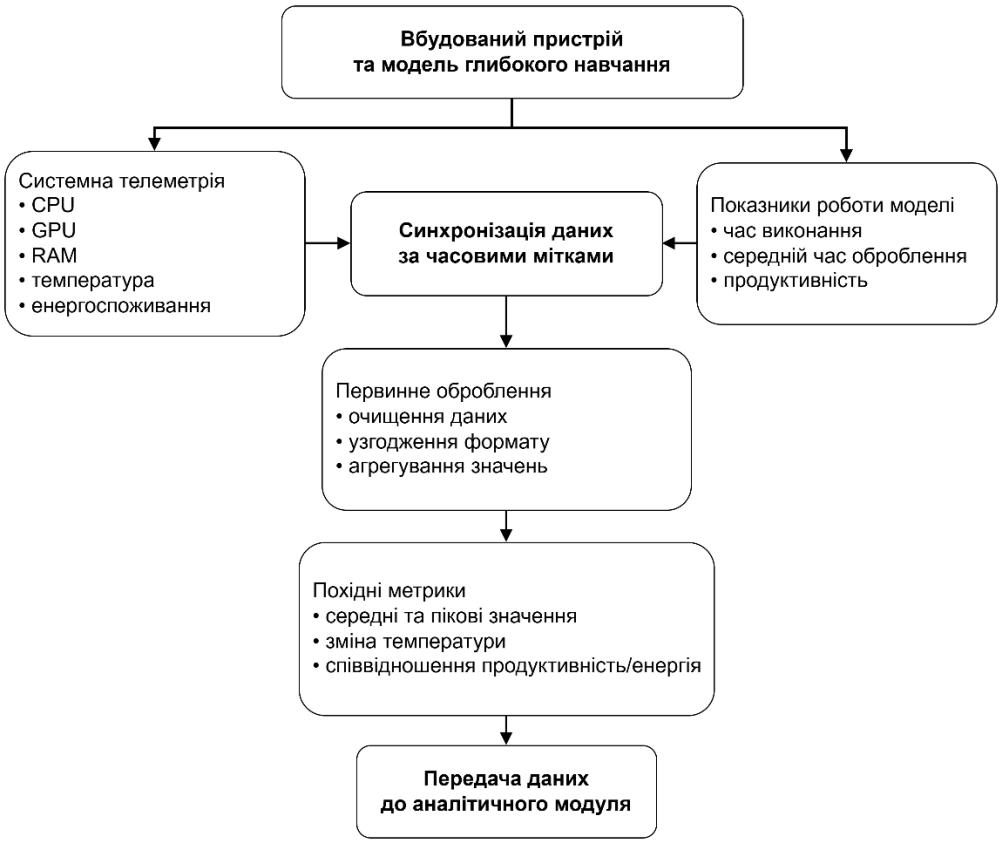

Рисунок 2.2 – Структурна схема збору даних та оброблення показників роботи моделі

На рисунку 2.2 відображено послідовність руху даних у межах розробленого модуля. Початковою ланкою визначено вбудований пристрій із запущеною моделлю глибокого навчання. Від нього одночасно надходять два потоки даних: системні дані та показники роботи моделі. Далі виконано синхронізацію цих потоків за часовими мітками, після чого здійснено первинне оброблення й обчислення похідних метрик. Підготовлені результати передано до аналітичного модуля, у якому формується узагальнена оцінка ефективності роботи моделі.

У результаті розроблення підсистеми збору даних забезпечено цілісне поєднання системних і часових показників у межах єдиної інформаційної структури. Це дало змогу підготувати дані до подальшого оцінювання продуктивності та енергоефективності моделі на вбудованій платформі, а також забезпечило інформаційну основу для формування рекомендацій щодо підвищення ефективності її роботи.

2.3 Моделі та методи аналізу продуктивності й енергоефективності

У межах проєктування інтелектуального модуля моніторингу сформовано набір моделей і методів аналізу, які забезпечили комплексне оцінювання роботи моделей глибокого навчання на вбудованому обчислювальному пристрої. Основу такого підходу побудовано на одночасному врахуванні часових показників, характеристик використання апаратних ресурсів і параметрів енергоспоживання. Це дало змогу перейти від ізольованого вимірювання окремих метрик до узагальненого оцінювання ефективності роботи моделі в реальних умовах функціонування [1, 3, 5].

Для аналізу продуктивності сформовано модель оцінювання, у якій ключовими показниками визначено час виконання моделі, середній час оброблення серії вхідних даних та продуктивність системи. Час виконання моделі відобразив тривалість одного циклу оброблення, а середній час оброблення дозволив оцінити стабільність роботи під час багаторазових запусків. Продуктивність охарактеризувала здатність системи обробляти певну кількість вхідних даних за одиницю часу. Сукупне використання цих показників забезпечило більш повне уявлення про швидкодію моделі на вбудованій платформі [1, 15, 34].

Окремо до моделі продуктивності включено показники використання апаратних ресурсів. До них віднесено завантаження центрального процесора, графічного прискорювача та використання оперативної пам’яті. Аналіз цих показників дав змогу встановити, який саме компонент системи визначає обмеження продуктивності під час виконання моделі. Наприклад, високий час виконання за помірного завантаження GPU може вказувати на неефективну організацію подання даних, тоді як постійно високий рівень використання пам’яті свідчить про надмірну ресурсомісткість моделі. Унаслідок цього оцінювання продуктивності доповнено аналізом причин виникнення вузьких місць [15, 17, 18].

Для оцінювання енергоефективності використано модель, у якій враховано середній рівень енергоспоживання, зміну температури пристрою та співвідношення між досягнутою продуктивністю і витраченими ресурсами. Такий підхід дозволив оцінити не лише швидкість роботи моделі, а й ефективність

використання апаратної платформи з погляду енергетичних витрат. У розробленій моделі враховано, що висока швидкодія не завжди означає високу енергоефективність, оскільки швидке виконання може супроводжуватися істотним зростанням потужності та теплового навантаження [5, 17, 27]. Саме тому до аналізу включено не один параметр, а сукупність взаємопов’язаних показників.

У межах оброблення даних сформовано систему похідних метрик, які спростили порівняння моделей і режимів їх роботи. До таких метрик віднесено середнє значення продуктивності, середнє енергоспоживання, приріст температури під час виконання, а також узагальнене співвідношення між продуктивністю та енергоспоживанням. Сформовані похідні характеристики дали змогу перейти від набору розрізнених вимірювань до порівнюваних числових оцінок, придатних для аналітичного висновку щодо ефективності роботи моделі на вбудованому пристрої.

Для практичної реалізації аналізу використано методи агрегування, статистичного узагальнення та порівняльного оцінювання. На їх основі для кожного показника визначено середні, мінімальні та максимальні значення, а також рівень коливання у часі. Це дозволило оцінити не лише середній режим функціонування, а й наявність нестабільних станів, які можуть проявлятися у вигляді різких піків навантаження, зростання температури або коливання часу виконання. Таким чином, аналіз доповнено елементами контролю стійкості роботи системи.

Окрему увагу приділено методу виявлення неефективних режимів функціонування. У межах цього методу визначено набір правил, за якими система інтерпретує стан моделі як проблемний. До таких станів віднесено перевищення допустимого температурного рівня, надмірне завантаження ресурсів, нестабільний час виконання та невідповідність між високим енергоспоживанням і низькою продуктивністю. Використання правил порогового аналізу спростило автоматизоване виявлення ситуацій, за яких обрана модель або режим її виконання є неефективними для конкретної вбудованої платформи [5, 17, 27].

У підсистемі аналізу також сформовано основу для порівняння кількох моделей або кількох режимів виконання однієї моделі. Такий підхід дав змогу зіставляти результати стандартного запуску та оптимізованого запуску, а також

оцінювати поведінку різних архітектур у межах однакових умов експерименту. Унаслідок цього аналіз продуктивності й енергоефективності набув не лише описового, а й порівняльного характеру, що є важливим для подальшого формування рекомендацій щодо вибору більш ефективної конфігурації [1, 12, 32].

Отже, у межах підрозділу сформовано моделі та методи аналізу, що охоплюють оцінювання швидкодії, використання апаратних ресурсів, температурного стану та енергоефективності. Розроблений підхід забезпечив комплексне оцінювання роботи моделей глибокого навчання на вбудованому обчислювальному пристрої та створив аналітичну основу для подальшого виявлення неефективних режимів функціонування і формування рекомендацій щодо підвищення ефективності роботи системи.

2.4 Алгоритм оцінювання стану моделі та формування рекомендацій

У межах проєктування інтелектуального модуля моніторингу розроблено алгоритм оцінювання стану моделі та формування рекомендацій. Алгоритм забезпечив перехід від накопичення даних до їх інтерпретації та дав змогу автоматично визначати, наскільки ефективно модель працює на вбудованому обчислювальному пристрої.

В основу алгоритму покладено аналіз трьох груп показників:

– показники швидкодії;
– показники використання апаратних ресурсів;
– показники енергоефективності.

До показників швидкодії віднесено час виконання моделі та продуктивність. До показників використання ресурсів включено завантаження CPU, GPU та обсяг використаної пам’яті. До показників енергоефективності віднесено температуру пристрою, енергоспоживання та похідне співвідношення між продуктивністю і витраченими ресурсами [1, 5, 17].

Алгоритм побудовано у вигляді послідовності кроків (рисунок 2.3).

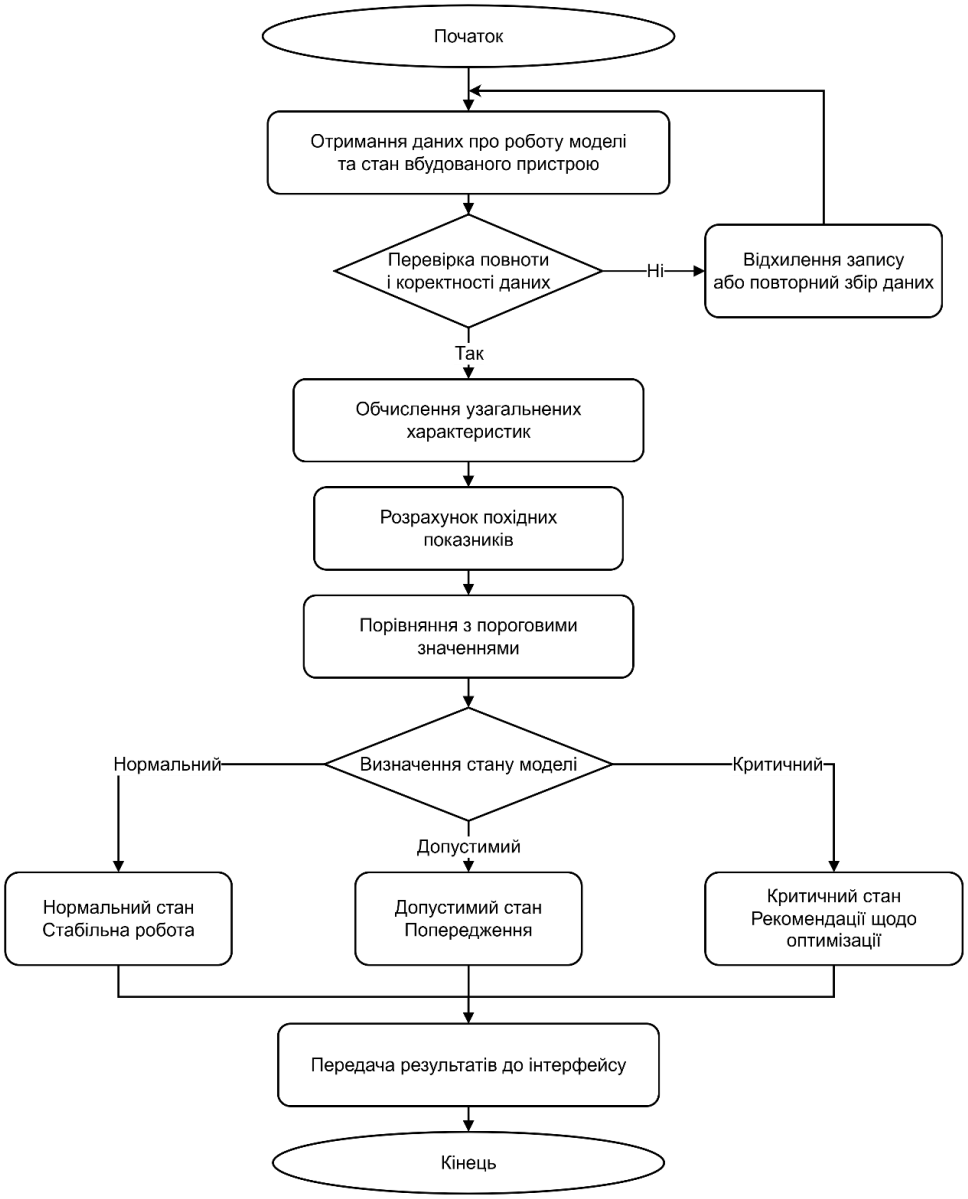

Рисунок 2.3 – Схема алгоритму оцінювання стану моделі та формування рекомендацій

Крок 1. Отримання вхідних даних. На початковому етапі алгоритм приймає з модуля збору даних значення системних і часових показників. До вхідного набору включено час виконання моделі, продуктивність, завантаження CPU і GPU, використання оперативної пам’яті, температуру та енергоспоживання.

Крок 2. Перевірка повноти та коректності даних. На цьому етапі перевіряється наявність усіх необхідних показників. Якщо частина даних відсутня або має некоректний формат, запис не використовується для підсумкового оцінювання. У результаті цього кроку забезпечується коректність подальшого аналізу.

Крок 3. Обчислення узагальнених характеристик. Для набору отриманих даних обчислюються середні значення часу виконання, продуктивності, навантаження на процесорні ресурси, температури та енергоспоживання. Додатково визначаються мінімальні та максимальні значення показників. Це дало змогу перейти від окремих вимірювань до узагальненої оцінки стану моделі.

Крок 4. Розрахунок похідних показників. На цьому етапі формуються похідні метрики, які використовуються для порівняння режимів роботи. До них віднесено співвідношення між продуктивністю та енергоспоживанням, зміну температури в процесі виконання, а також узагальнений показник ефективності.

Крок 5. Порівняння з пороговими значеннями. Отримані характеристики порівнюються з наперед заданими межами допустимих значень. Наприклад, перевіряється, чи не перевищено допустиму температуру, чи не є час виконання нестабільним, чи не спостерігається надмірне навантаження на ресурси та чи відповідає досягнута продуктивність витраченій енергії.

Крок 6. Визначення стану моделі. На основі порівняння показників із порогами встановлюється один із трьох станів моделі:

– нормальний – усі основні показники перебувають у допустимих межах;

– допустимий – окремі показники мають незначні відхилення, але загальна робота моделі залишається стабільною;

– критичний – зафіксовано суттєве перевищення порогових значень або поєднання кількох негативних ознак.

Крок 7. Формування рекомендацій. Після визначення стану моделі алгоритм формує рекомендацію. Якщо виявлено перевантаження ресурсів, система пропонує використання легшої моделі або зменшення розміру вхідних даних. Якщо зафіксовано надмірне енергоспоживання, формується рекомендація щодо переходу до оптимізованого режиму виконання. Якщо виявлено нестабільність часових показників, рекомендується додаткове налаштування середовища або використання оптимізованої версії моделі.

Крок 8. Передача результату до інтерфейсу. На завершальному етапі підсумковий стан моделі, розраховані показники та сформовані рекомендації передаються до інтерфейсу подання результатів. У такий спосіб користувач

отримує не лише числові значення, а й узагальнений висновок щодо ефективності роботи моделі.

Отже, розроблений алгоритм забезпечив послідовне оцінювання стану моделі на основі системних, часових та енергетичних показників. Це дало змогу автоматизувати виявлення неефективних режимів роботи та сформувати основу для практичної підтримки прийняття рішень щодо підвищення продуктивності й енергоефективності.

2.5 Проєктування інформаційного забезпечення

У результаті проєктування інформаційного забезпечення сформовано структуру даних інтелектуального модуля моніторингу, яка складається з п’яти основних інформаційних об’єктів: модель, запуск, системні дані, метрики, рекомендації. Така структура забезпечила впорядковане збереження даних про модель, параметри її виконання, стан вбудованого пристрою, результати обчислення показників та підсумкові висновки системи.

1. Інформаційний об’єкт «Модель» – призначено для збереження загальних відомостей про модель глибокого навчання, яка використовується у модулі моніторингу. Поля інформаційного об’єкту «Модель»:

– model_id – унікальний ідентифікатор моделі;
– model_name – назва моделі;
– model_type – тип або архітектура моделі;
– model_format – формат подання моделі;
– input_size – розмір вхідних даних;
– optimization_mode – режим виконання моделі.

2. Інформаційний об’єкт «Запуск» – описує окремий цикл виконання моделі на вбудованому пристрої. Поля інформаційного об’єкту «Запуск»:

– run_id – унікальний ідентифікатор запуску;
– model_id – посилання на модель;
– start_time – час початку виконання;
– end_time – час завершення виконання;

– input_batch – кількість вхідних даних у запуску;

– run_status – стан завершення запуску.

3. Інформаційний об’єкт «Системні дані» – призначено для збереження показників стану вбудованого пристрою під час виконання моделі. Поля інформаційного об’єкту «Системні дані»:

– system_id – унікальний ідентифікатор запису;

– run_id – посилання на запуск;

– timestamp – час фіксації даних;

– cpu_load – завантаження CPU, %;

– gpu_load – завантаження GPU, %;

– ram_usage – використання оперативної пам’яті, МБ або %;

– temperature – температура пристрою, °C;

– power_value – поточне енергоспоживання.

4. Інформаційний об’єкт «Метрики» – містить узагальнені результати оброблення даних для кожного запуску моделі. Поля інформаційного об’єкту «Метрики»:

– metric_id – унікальний ідентифікатор метрик;

– run_id – посилання на запуск;

– avg_execution_time – середній час виконання;

– min_execution_time – мінімальний час виконання;

– max_execution_time – максимальний час виконання;

– performance_value – продуктивність;

– avg_cpu_load – середнє завантаження CPU;

– avg_gpu_load – середнє завантаження GPU;

– avg_ram_usage – середнє використання пам’яті;

– avg_temperature – середня температура;

– avg_power_value – середнє енергоспоживання;

– efficiency_index – інтегральний показник ефективності.

5. Інформаційний об’єкт «Рекомендації» – зберігає результати аналітичного оцінювання стану моделі. Поля інформаційного об’єкту «Рекомендації»:

– recommendation_id – унікальний ідентифікатор рекомендації;

– run_id – посилання на запуск;

– state_name – визначений стан моделі;

– problem_type – виявлена проблема;

– recommendation_text – текст рекомендації;

– created_at – час формування рекомендації.

Структура даних інтелектуального модуля моніторингу представлена в таблиці 2.2 та на рисунку 2.4.

Таблиця 2.2 – Структура даних інтелектуального модуля моніторингу

Інформаційний об’єкт	Основні поля	Призначення
Модель	model_id, model_name, model_type, model_format, input_size, optimization_mode	Збереження відомостей про модель
Запуск	run_id, model_id, start_time, end_time, input_batch, run_status	Опис окремого виконання моделі
Системні дані	system_id, run_id, timestamp, cpu_load, gpu_load, ram_usage, temperature, power_value	Збереження даних про стан пристрою
Метрики	metric_id, run_id, avg_execution_time, performance_value, avg_cpu_load, avg_gpu_load, avg_ram_usage, avg_temperature, avg_power_value, efficiency_index	Збереження обчислених показників
Рекомендації	recommendation_id, run_id, state_name, problem_type, recommendation_text, created_at	Збереження висновків і рекомендацій

У результаті проєктування інформаційного забезпечення сформовано структуру даних інтелектуального модуля моніторингу, яка охопила п’ять основних інформаційних об’єктів: «Модель», «Запуск», «Системні дані», «Метрики» та «Рекомендації». Для кожного визначено склад полів, що забезпечило впорядковане збереження відомостей про модель, параметри її виконання, стан вбудованого пристрою, результати оброблення показників і сформовані рекомендації.

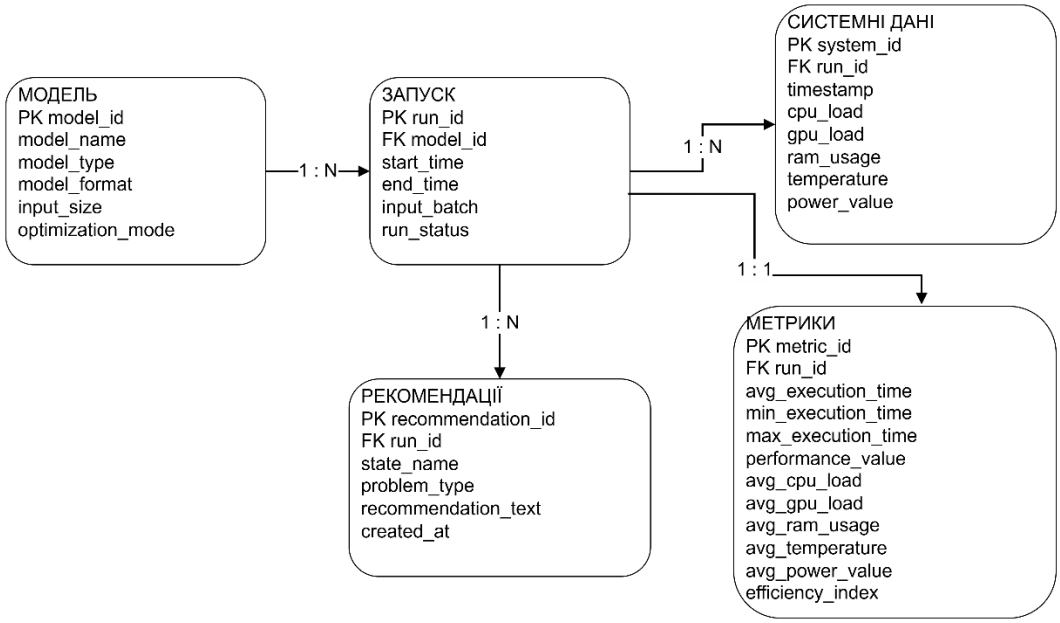

Рисунок 2.4 – Структура даних інтелектуального модуля моніторингу

Розроблена структура даних забезпечила логічний зв’язок між етапами збору, оброблення та аналітичного оцінювання інформації в межах інтелектуального модуля моніторингу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ МОДУЛЯ

3.1 Реалізація програмного прототипу модуля моніторингу

У межах даної роботи реалізовано програмний прототип інтелектуального модуля моніторингу продуктивності та енергоефективності моделей глибокого навчання на вбудованому обчислювальному пристрої. Прототип реалізовано як модульну програмну систему, що забезпечила запуск моделі, збір системних даних, оброблення показників і формування підсумкових результатів.

Структуру програмного прототипу побудовано з кількох взаємопов’язаних компонентів: модуля запуску моделі, модуля збору системних даних, модуля оброблення показників та модуля формування результатів (див. рисунок 2.1). В таблиці 3.1 подано призначення основних компонентів програмного прототипу модуля моніторингу.

Таблиця 3.1 – Основні програмні модулі прототипу

Модуль	Призначення
Модуль запуску моделі	Завантаження моделі та керування її виконанням
Модуль збору даних	Зчитування системних показників пристрою
Модуль оброблення	Узагальнення та аналіз отриманих значень
Модуль формування результатів	Виведення підсумкових метрик і рекомендацій

Основним середовищем реалізації обрано Python, що забезпечило сумісність із бібліотеками глибокого навчання та засобами опрацювання даних. Для роботи з моделями використано PyTorch і TorchVision. У програмному прототипі реалізовано окрему функцію завантаження моделі, яка вибирає потрібну архітектуру та переводить її в режим виконання. Фрагмент відповідної реалізації наведено в лістингу 3.1.

```
import torch
from torchvision import models

def load_model(model_name: str):
    if model_name == "mobilenet_v2":
        model = models.mobilenet_v2(weights="DEFAULT")
    elif model_name == "resnet18":
        model = models.resnet18(weights="DEFAULT")
```

```
    else:
        raise ValueError("Невідома модель")
    model.eval()
    return model
```

Лістинг 3.1 – Фрагмент завантаження моделі

Для оцінювання швидкодії реалізовано вимірювання часу виконання моделі. Перед запуском формувався вхідний тензор, після чого фіксувався час початку та завершення оброблення. Це дало змогу визначити тривалість одного запуску моделі. Відповідний фрагмент наведено в лістингу 3.2.

```
import time
import torch

def measure_execution_time(model, input_tensor):
    start_time = time.time()
    with torch.no_grad():
        _ = model(input_tensor)
    end_time = time.time()
    return end_time - start_time
```

Лістинг 3.2 – Фрагмент вимірювання часу виконання моделі

Паралельно з виконанням моделі реалізовано збір системних даних пристрою. Для цього використано зчитування параметрів завантаження процесорних ресурсів, пам’яті та температури. Отримані значення зберігалися у структурованому вигляді для подальшого аналізу. Приклад фрагмента збору даних наведено в лістингу 3.3.

```
import psutil

def get_system_data():
    return {
        "cpu_load": psutil.cpu_percent(),
        "ram_usage": psutil.virtual_memory().percent
    }
```

Лістинг 3.3 – Фрагмент збору системних даних

Після накопичення показників реалізовано їх узагальнення. Для цього обчислювалися середні значення часу виконання, завантаження ресурсів та інших контрольованих характеристик. Такий підхід забезпечив перехід від окремих вимірювань до зведених показників, придатних для подальшого оцінювання. Приклад фрагмента оброблення наведено в лістингу 3.4.

```
def calculate_average(values):
    if not values:
        return 0.0
    return sum(values) / len(values)
```

Лістинг 3.4 – Фрагмент обчислення середнього значення

Структура програмного проєкту модуля моніторингу представлено на рисунку 3.1.

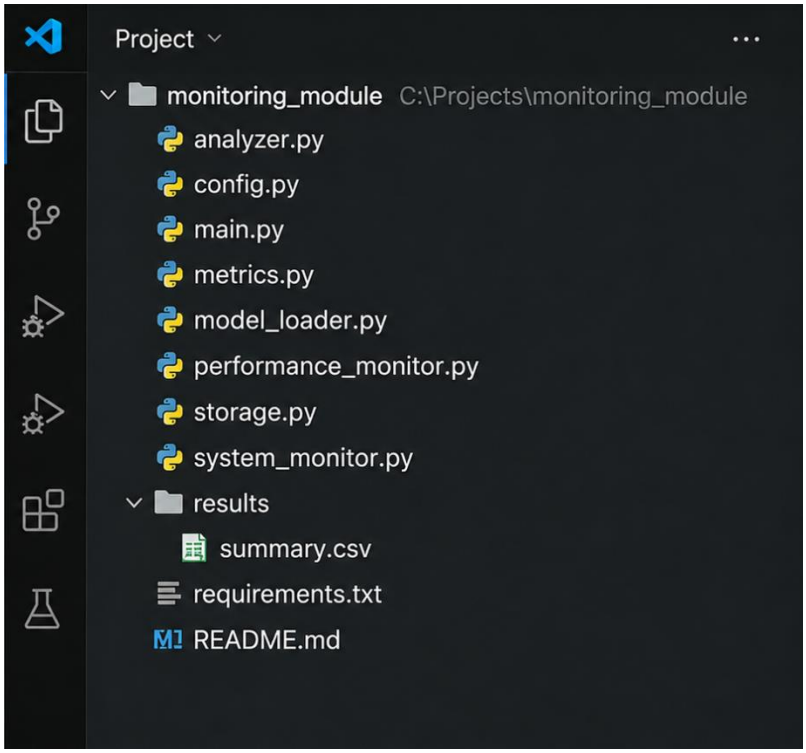

Рисунок 3.1 – Структура програмного проєкту модуля моніторингу

Для експериментального дослідження реалізовано механізм завантаження кількох моделей глибокого навчання, зокрема MobileNetV2, ResNet18, ShuffleNet V2 та EfficientNet-B0. Це дало змогу виконати порівняння моделей за однакових умов запуску. Фрагмент реалізації завантаження моделей наведено на рисунку 3.2.

На рисунку 3.2 показано, що вибір моделі здійснюється за її назвою. Після завантаження модель переводиться в режим оцінювання, що забезпечує коректне виконання без етапу навчання.

Повні лістинги реалізованого програмного забезпечення наведено в додатку А, де подано завершену структуру коду модуля моніторингу, функції запуску моделей, збирання даних, оброблення метрик та формування результатів.

model_loader.py

```python
import torch
from torchvision import models

def load_model(model_name: str):
    if model_name == "mobilenet_v2":
        model = models.mobilenet_v2(weights="DEFAULT")
    elif model_name == "resnet18":
        model = models.resnet18(weights="DEFAULT")
    elif model_name == "shufflenet_v2":
        model = models.shufflenet_v2_x1_0(weights="DEFAULT")
    elif model_name == "efficientnet_b0":
        model = models.efficientnet_b0(weights="DEFAULT")
    else:
        raise ValueError("Невідома модель")
    model.eval()
    return model

def get_device():
    return torch.device('cuda' if torch.cuda.is_available() else 'cpu')


def create_input_tensor(batch_size=1, channels=3, height=224, width=224):
    return torch.randn(batch_size, channels, height, width)
```

Рисунок 3.2 – Прикллад реалізації завантаження моделей у програмному середовищі

У результаті реалізації програмного прототипу підтверджено можливість поєднання запуску моделей глибокого навчання, збору системних даних і розрахунку показників ефективності в межах єдиної програмної системи.

3.2 Реалізація підсистеми збору метрик продуктивності та енергоспоживання

У межах програмного прототипу реалізовано підсистему збору даних, призначену для фіксації показників продуктивності моделі та параметрів стану вбудованого обчислювального пристрою під час її виконання. Реалізована підсистема забезпечила автоматичне одержання часових, ресурсних і енергетичних характеристик, що сформувало інформаційну основу для подальшого аналітичного оцінювання роботи моделі.

Підсистему побудовано за принципом паралельного спостереження за двома групами показників. До першої групи віднесено показники роботи моделі: час виконання, кількість запусків, середній час оброблення та продуктивність. До другої групи віднесено системні показники пристрою: завантаження CPU,

завантаження GPU, використання оперативної пам’яті, температура та енергоспоживання [16–18, 34]. Такий підхід забезпечив одночасне спостереження як за поведінкою моделі, так і за станом апаратної платформи.

На рисунку 3.3 представлено структуру реалізованої підсистеми збору даних. У її складі виділено блок ініціалізації моніторингу, блок періодичного зчитування системних показників, блок фіксації часових параметрів виконання моделі та блок збереження результатів у структурованому вигляді. Побудована схема забезпечила узгоджену послідовність збору та накопичення даних у межах одного циклу запуску моделі.

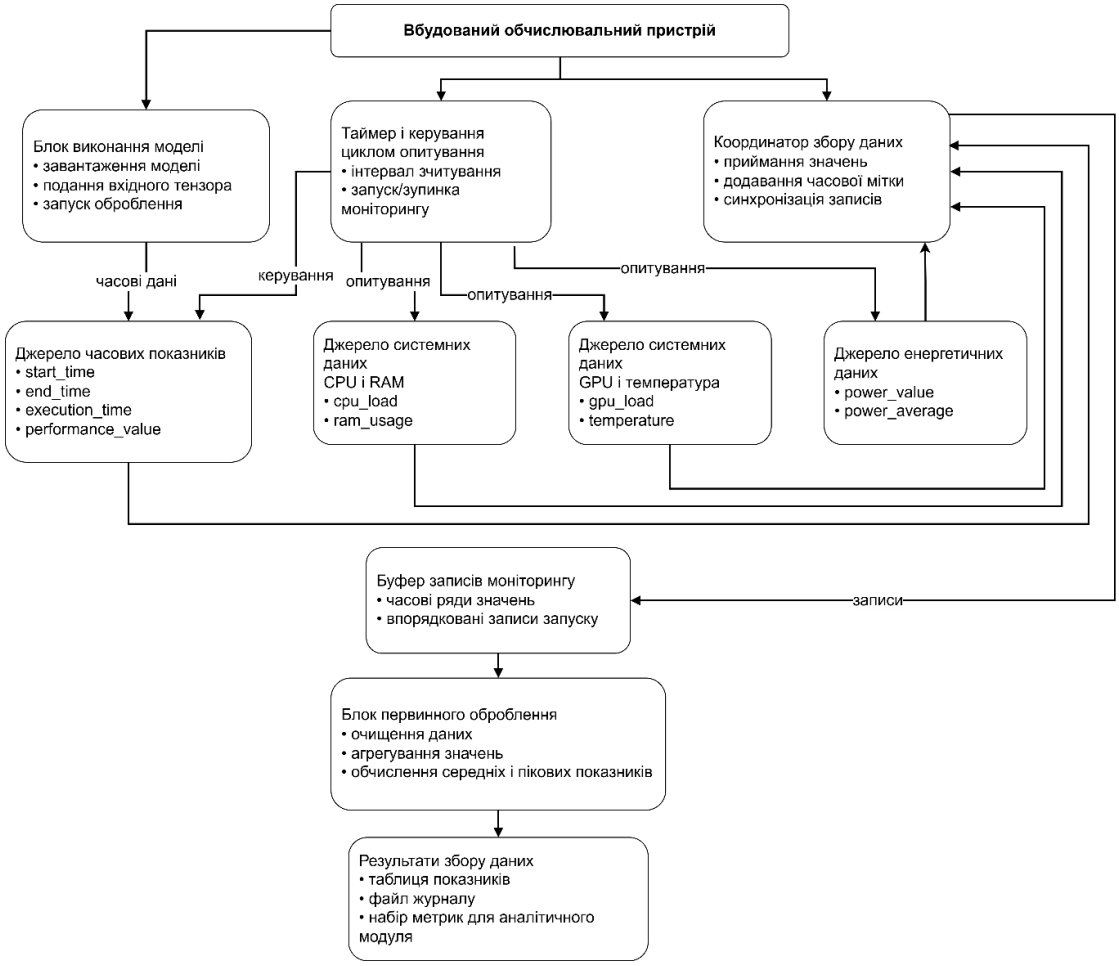

Рисунок 3.3 – Структура підсистеми збору даних продуктивності та енергоспоживання

Часові показники роботи моделі зафіксовано безпосередньо в модулі запуску. Для цього перед початком виконання моделі записувався момент старту, а після завершення – момент завершення. Різниця між цими значеннями визначала

тривалість одного запуску. Для серії запусків обчислювався середній час виконання, що дало змогу оцінити стабільність роботи моделі. Фрагмент відповідної реалізації наведено в лістингу 3.5.

```
import time
import torch

def measure_execution_time(model, input_tensor):
    start_time = time.time()
    with torch.no_grad():
        _ = model(input_tensor)
    end_time = time.time()
    return end_time - start_time
```

Лістинг 3.5 – Фрагмент вимірювання часу виконання моделі

Системні показники пристрою зчитувались у фоновому режимі через визначені часові інтервали. У реалізованому прототипі забезпечено одержання даних про завантаження центрального процесора та використання оперативної пам’яті засобами програмного доступу до системних ресурсів. Для вбудованої платформи сімейства NVIDIA Jetson передбачено можливість підключення даних про GPU, температуру та енергоспоживання через системні засоби моніторингу [17]. Приклад фрагмента збору базових системних показників наведено в лістингу 3.6.

```
import psutil

def get_system_data():
    return {
        "cpu_load": psutil.cpu_percent(),
        "ram_usage": psutil.virtual_memory().percent
    }
```

Лістинг 3.6 – Фрагмент збору системних показників

Для накопичення результатів реалізовано збереження кожного вимірювання у вигляді структурованого запису. Кожний запис містив часову мітку, значення системних показників та параметри поточного запуску моделі. Це забезпечило формування впорядкованого набору даних для подальшого узагальнення. Після завершення серії запусків виконувалось агрегування значень і розрахунок середніх показників продуктивності та використання ресурсів.

У таблиці 3.2 подано склад основних показників, що збираються підсистемою моніторингу.

Таблиця 3.2 – Основні показники підсистеми збору даних

Група показників	Показник	Призначення
Часові	Час виконання моделі	Оцінювання швидкодії одного запуску
Часові	Середній час виконання	Узагальнене оцінювання серії запусків
Продуктивність	Кількість оброблень за одиницю часу	Оцінювання пропускної здатності
Ресурсні	Завантаження CPU	Оцінювання навантаження на центральний процесор
Ресурсні	Завантаження GPU	Оцінювання використання графічного прискорювача
Ресурсні	Використання RAM	Оцінювання споживання пам’яті
Теплові	Температура пристрою	Контроль теплового режиму
Енергетичні	Енергоспоживання	Оцінювання витрат ресурсів під час виконання

Реалізовану підсистему орієнтовано на мінімізацію власних накладних витрат. Для цього період зчитування системних показників вибрано таким чином, щоб не створювати надмірного навантаження на пристрій. Оброблення накопичених значень перенесено на завершальний етап циклу запуску, що зменшило вплив процесу моніторингу на результати вимірювання. Унаслідок цього забезпечено коректність зібраних показників та придатність їх до подальшого аналізу.

У процесі виконання моделі модуль моніторингу виводить у термінал основні показники поточного запуску: назву моделі, номер ітерації, час виконання, завантаження CPU та GPU, використання пам’яті, температуру та потужність. Приклад запуску модуля моніторингу наведено на рисунку 3.4.

Як видно з рисунка 3.4, під час кожного запуску фіксуються як часові показники виконання моделі, так і системні дані пристрою. Це забезпечує зв’язок між продуктивністю моделі та фактичним станом обчислювальної платформи.

Для подальшого аналізу результати моніторингу збережено у структурованому CSV-файлі. У ньому зафіксовано назву моделі, середній час виконання, показник продуктивності, середнє завантаження CPU і GPU, температуру, потужність, індекс ефективності та підсумкову оцінку стану.

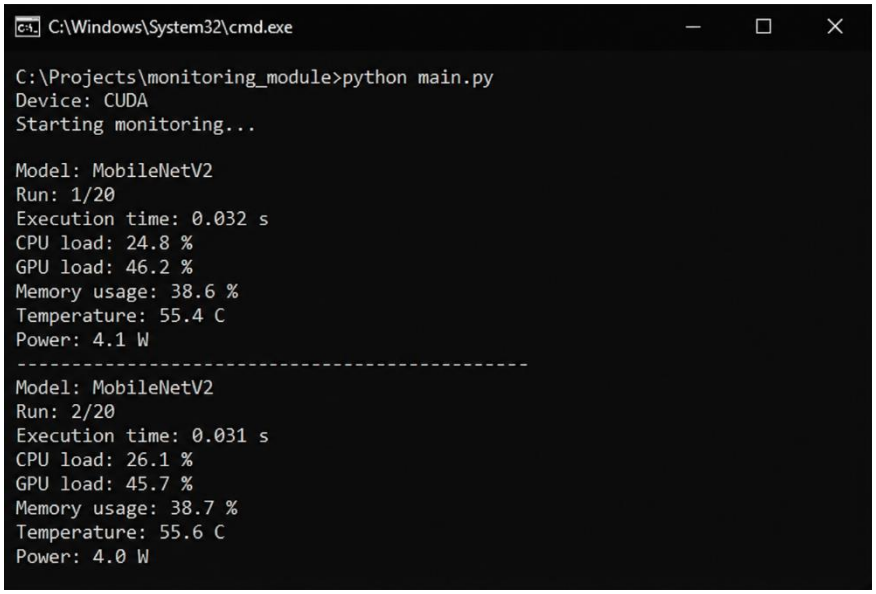

Рисунок 3.4 – Приклад запуску модуля моніторингу в терміналі

Приклад збереження результатів наведено на рисунку 3.5.

summary.csv

	A	B	C	D	E	F	G	H	J
1	model_name	avg_execution_time	performance_value	avg_cpu_load	avg_gpu_load	avg_temperature	avg_power_value	efficiency_index	state_name
2	MobileNetV2	0.0318	31.43	24.82	46.21	55.48	4.12	7.62	Good
3	ResNet18	0.0924	11.03	32.79	68.34	63.21	7.43	3.25	Average
4	ShuffleNet V2	0.0283	35.34	23.11	42.17	53.72	3.86	9.16	Excellent
5	EfficientNet-B0	0.0838	11.93	29.64	57.28	61.18	6.61	4.25	Good
6									

summary

Рисунок 3.5 – Приклад збереження результатів моніторингу у файлі summary.csv

Рисунок 3.5 показує, що результати подано у табличному вигляді, що спростило їх подальше порівняння та використання для побудови графіків.

Таким чином, у підсистемі збору даних реалізовано механізми фіксації часу виконання моделі, періодичного зчитування системних показників та структурованого збереження результатів.

3.3 Інтеграція модуля з моделями глибокого навчання та платформою

У межах реалізації програмного прототипу виконано інтеграцію розробленого модуля моніторингу з моделями глибокого навчання та вбудованою обчислювальною платформою. Інтеграцію побудовано так, щоб запуск моделі, зчитування системних показників, оброблення даних та формування результатів здійснювалися в межах єдиного програмного контуру. Це забезпечило узгоджену роботу всіх компонентів системи та створило основу для експериментального оцінювання продуктивності й енергоефективності моделей на вбудованому пристрої [3, 12, 15].

В якості вбудованої обчислювальної платформи використано NVIDIA Jetson Nano Developer Kit (рисунок 3.6), що є компактним одноплатним пристроєм для запуску моделей глибокого навчання на edge-рівні. Платформа має графічний прискорювач NVIDIA Maxwell із 128 CUDA-ядрами, чотириядерний ARM Cortex-A57, 4 ГБ оперативної пам’яті та підтримує роботу з CUDA, cuDNN, TensorRT і JetPack SDK. Використання Jetson Nano забезпечило можливість дослідження продуктивності та енергоефективності моделей глибокого навчання в умовах обмежених апаратних ресурсів [16–18].

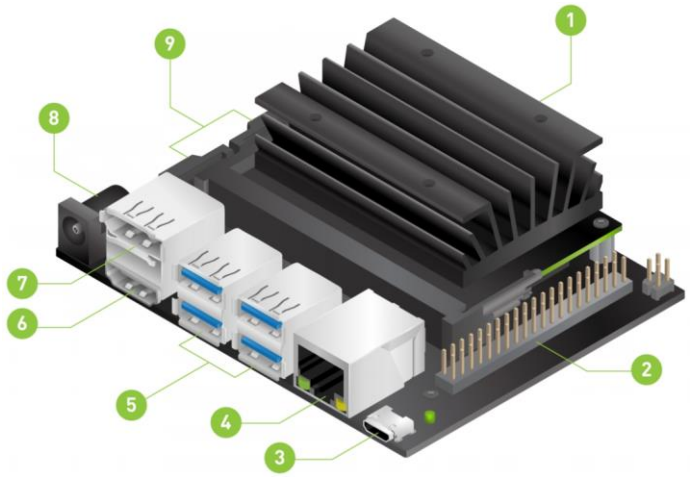

Рисунок 3.6 – Будова NVIDIA Jetson Nano Developer Kit: 1 - слот для картки microSD як основного сховища даних; 2 - 40-контактний роз’єм розширення; 3 - порт Micro-USB для вхідного живлення 5 В або для режиму пристрою; 4-порт Gigabit Ethernet; 5 - порти USB 3.0 — 4 шт.; 6 - вихідний порт HDMI; 7 - роз’єм DisplayPort; 8 - роз’єм живлення DC Barrel jack для вхідного живлення 5 В.

1) роз’єми камери MIPI CSI-2.

Характеристики платформи, наведені в таблиці 3.3, показують, що NVIDIA Jetson Nano має обмежені ресурси порівняно із серверними GPU-системами. Саме тому ця платформа використана як базове середовище для перевірки роботи модуля моніторингу та оцінювання моделей глибокого навчання в умовах edge-обчислень.

Таблиця 3.3 – Характеристики використаної вбудованої платформи

Характеристика	Значення
Платформа	NVIDIA Jetson Nano Developer Kit
GPU	NVIDIA Maxwell, 128 CUDA-ядер
CPU	Quad-core ARM Cortex-A57
Оперативна пам’ять	4 ГБ LPDDR4
Програмне середовище	JetPack SDK, CUDA, cuDNN, TensorRT, Python, PyTorch
Клас пристрою	Edge/embedded AI-платформа
Призначення в роботі	Виконання моделей глибокого навчання та збір показників продуктивності й енергоефективності

Програмну інтеграцію виконано на основі Python із використанням бібліотек PyTorch і TorchVision для роботи з моделями, а також засобів доступу до системних показників і результатів виконання [12, 20, 21]. Такий вибір забезпечив сумісність із сучасними архітектурами моделей та спростив підключення модуля моніторингу до процесу їх виконання.

У програмному рішенні реалізовано механізм підключення кількох моделей глибокого навчання в межах єдиного сценарію запуску. Для експериментального використання інтегровано моделі MobileNetV2, ResNet18, ShuffleNet V2 та EfficientNet-B0, що дало змогу порівнювати поведінку архітектур із різною структурою та різним рівнем ресурсомісткості. Для всіх моделей забезпечено однаковий порядок завантаження, підготовки вхідного тензора, запуску виконання та збереження результатів, що сформувало єдині умови інтеграції.

На рисунку 3.7 представлено схему інтеграції модуля моніторингу з моделями глибокого навчання та вбудованою платформою. У побудованій схемі поєднано блок вибору моделі, середовище її виконання, підсистему збору даних, модуль оброблення та блок формування результатів. Така організація забезпечила

безперервне отримання показників роботи моделі без розриву між етапами запуску та аналітичного оцінювання.

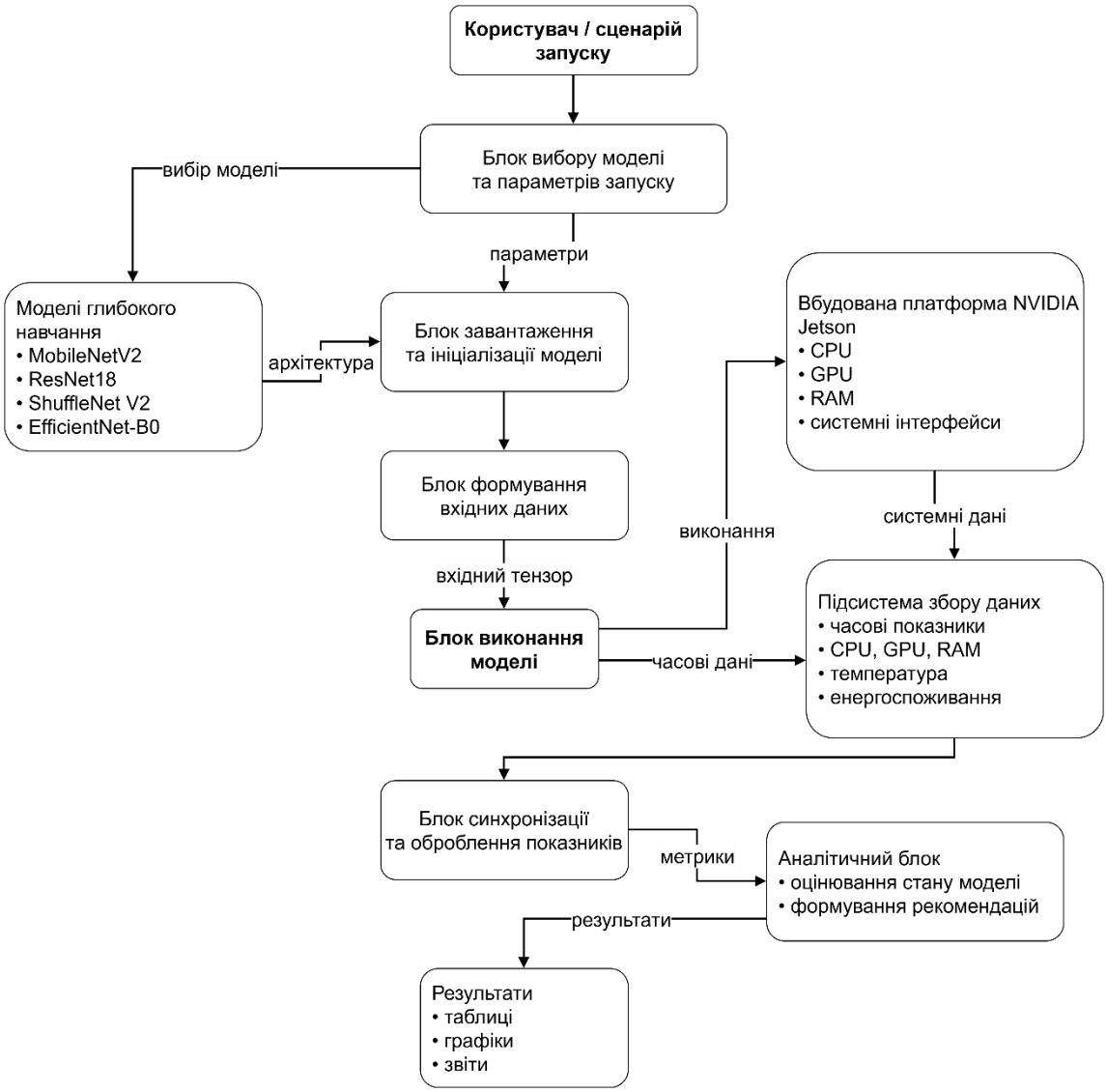

Рисунок 3.7 – Схема інтеграції модуля моніторингу з моделями глибокого навчання та вбудованою платформою

Інтеграцію з моделями реалізовано через окремий блок завантаження архітектури, у якому назва моделі використовується як параметр вибору потрібного сценарію ініціалізації. Після завантаження модель переводиться в режим виконання та розміщується на вибраному обчислювальному пристрої. Далі формується вхідний тензор заданого розміру, який подається на вхід моделі. Такий підхід забезпечив уніфікований механізм підключення моделей різних типів і спростив їх подальше порівняння в межах одного програмного середовища.

Інтеграцію із вбудованою платформою виконано паралельно з інтеграцією

моделей. Під час виконання моделі модуль моніторингу зчитує завантаження CPU, GPU, використання оперативної пам’яті, температуру та енергоспоживання. Одночасно фіксуються час виконання моделі та продуктивність. Усі отримані значення позначаються часовими мітками й передаються до блоку оброблення. Це забезпечило узгодження системних і часових показників та дало змогу співвіднести зміну стану пристрою з конкретним запуском моделі.

У таблиці 3.4 подано основні компоненти інтеграції та їх призначення.

У ході інтеграції враховано необхідність мінімізації впливу самого модуля моніторингу на результати вимірювання. Для цього зчитування системних показників виконано у фоновому режимі, а частину оброблення перенесено на етап після завершення серії запусків. Це зменшило накладні витрати підсистеми моніторингу та підвищило коректність одержаних результатів. Унаслідок цього інтегрована система забезпечила стабільне виконання моделей і придатність зібраних даних до подальшого аналізу.

Таблиця 3.4 – Основні компоненти інтеграції модуля з моделями та платформою

Компонент	Призначення
Блок вибору моделі	Вибір архітектури та режиму запуску
Блок завантаження моделі	Ініціалізація моделі та переведення в режим виконання
Блок формування вхідних даних	Підготовка тензора заданого розміру
Блок виконання моделі	Запуск оброблення даних на пристрої
Блок збору системних даних	Зчитування CPU, GPU, RAM, температури, енергоспоживання
Блок синхронізації	Узгодження часових і системних показників
Блок оброблення результатів	Обчислення узагальнених показників
Блок подання результатів	Формування таблиць, файлів і підсумкових висновків

Окремо реалізовано можливість роботи як у стандартному режимі виконання моделі, так і в режимі використання підготовлених оптимізованих представлень. Це створило основу для подальшого порівняння звичайного запуску моделі та запуску з використанням оптимізованих засобів виконання, що є важливим для

експериментального дослідження продуктивності та енергоефективності.

Інтеграцію модуля з моделями глибокого навчання реалізовано через основний сценарій запуску, у якому сформовано список моделей для тестування та послідовно виконано моніторинг кожної з них (рисунок 3.8).

main.py

```
from config import AppConfig
from performance_monitor import run_monitoring

models_to_test = [
    'MobileNetV2',
    'ResNet18',
    'ShuffleNet_V2_X1_0',
    'EfficientNet-B0'
]
for model_name in models_to_test:
    print(f"\n{'-'*50}\nModel: {model_name} {'-'*50}")
    result = run_monitoring(model_name, runs=20)
    run_monitoring(model_name, runs=20)
    print(f"Efficiency index: {result['efficiency_index']:.2f} | State: {result['state_name']}")

```

Рисунок 3.8 – Інтеграція модуля з моделями глибокого навчання

Як показано на рисунку 3.8, для кожної моделі викликається функція моніторингу з однаковою кількістю запусків. Це забезпечило однакові умови експериментального порівняння та дало змогу отримати узагальнений індекс ефективності для кожної архітектури.

Таким чином, інтеграція модуля моніторингу з моделями глибокого навчання та вбудованою платформою забезпечила цілісне функціонування всієї програмної системи. У межах реалізованого рішення поєднано завантаження моделі, виконання оброблення, збір системних даних, обчислення показників і подання результатів.

3.4 Організація та методика проведення експериментів

У межах кваліфікаційної роботи організовано експериментальне дослідження, спрямоване на перевірку працездатності розробленого модуля моніторингу та оцінювання його здатності фіксувати показники продуктивності й

енергоефективності моделей глибокого навчання на вбудованому обчислювальному пристрої. Експериментальну частину побудовано так, щоб забезпечити однакові умови запуску моделей, коректність порівняння результатів та придатність зібраних даних до подальшого аналізу [1, 15, 23].

Основу експериментів сформовано на використанні вбудованої платформи класу NVIDIA Jetson, на якій реалізовано запуск моделей глибокого навчання та роботу програмного модуля моніторингу. У межах дослідження використано кілька моделей, що відрізняються архітектурою та рівнем ресурсомісткості, а саме MobileNetV2, ResNet18, ShuffleNet V2 та EfficientNet-B0 [8, 14, 25, 29]. Такий вибір забезпечив можливість порівняти поведінку як відносно легких, так і більш складних моделей в однаковому програмно-апаратному середовищі.

Для тестування модуля сформовано синтетичний набір вхідних даних, який складається із RGB-зображень розміром 224×224 пікселі. Такі дані відповідають типовому формату входу для моделей MobileNetV2, ResNet18, ShuffleNet V2 та EfficientNet-B0. Такий підхід забезпечив однакові вхідні умови для всіх моделей і дав змогу зосередити експеримент на аналізі продуктивності та енергоефективності, а не на особливостях конкретного набору зображень. Формування тестового набору даних наведено на рисунку 3.9.

metrics.py

```python
import torch

def create_test_dataset(num_samples: int = 20, batch_size: int = 1):
    dataset = []

    for i in range(num_samples):
        input_tensor = torch.randn(batch_size, 3, 224, 224)
        dataset.append(input_tensor)

    return dataset

# Приклад використання
if __name__ == "__main__":
    data = create_test_dataset()
    print(f"Кількість зразків: {len(data)}")
    print(f"Розмір одного тензора: {data[0].shape}")
```

C:\Windows\System32\cmd.exe

```
C:\Projects\monitoring_module>python metrics.py
Кількість зразків: 20
Розмір одного тензора: torch.Size([1, 3, 224, 224])
```

Рисунок 3.9 – Формування синтетичного тестового набору даних

Набір використано не для навчання або перевірки точності класифікації, а для багаторазового запуску моделей і фіксації показників продуктивності, використання ресурсів, температури та енергоспоживання.

У таблиці 3.5 подано характеристики тестового набору даних.

Таблиця 3.5 – Характеристика тестового набору даних

Показник	Значення
Тип даних	Синтетичні RGB-зображення
Розмір зображення	224×224 пікселі
Кількість каналів	3
Формат тензора	1×3×224×224
Призначення	Тестування продуктивності та енергоефективності
Використання для навчання	Не використовувався

Для всіх моделей сформовано єдині умови проведення експерименту. Розмір вхідних даних зафіксовано, кількість запусків для кожної моделі встановлено однаковою, а порядок отримання показників стандартизовано. Перед початком основної серії запусків виконано попередню ініціалізацію моделі, що дало змогу зменшити вплив стартових накладних витрат на підсумкові результати. Після цього для кожної моделі здійснено серію повторних запусків, у межах яких підсистема моніторингу фіксувала часові, ресурсні, температурні та енергетичні показники.

У таблиці 3.6 подано основні параметри експериментального дослідження.

Таблиця 3.6 – Основні параметри експериментів

Параметр	Значення
Цільова платформа	Вбудований пристрій класу NVIDIA Jetson
Середовище реалізації	Python, PyTorch, TorchVision
Досліджувані моделі	MobileNetV2, ResNet18, ShuffleNet V2, EfficientNet-B0
Режим запуску	Послідовні серії повторних виконань
Контрольовані показники	Час виконання, продуктивність, CPU, GPU, RAM, температура, енергоспоживання
Результат експерименту	Набір метрик для подальшого оцінювання ефективності

Методику проведення експериментів реалізовано у вигляді послідовності

кроків. На першому етапі обиралася модель та задавалися параметри запуску. На другому етапі виконувалося завантаження моделі й підготовка вхідних даних. На третьому етапі запускалася серія оброблень, у межах якої автоматично здійснювався збір системних показників. На четвертому етапі результати зберігалися у вигляді структурованих записів. На п’ятому етапі виконувалося узагальнення отриманих значень та формування підсумкових показників для кожної моделі. Така методика забезпечила однаковий порядок проведення експериментів для всіх варіантів дослідження.

Для підвищення достовірності результатів використано повторні запуски кожної моделі. Це дало змогу зменшити вплив випадкових коливань часу виконання, нестабільності системного навантаження та короткочасних змін температурного режиму. Підсумкові значення показників визначено як узагальнені характеристики серії запусків, зокрема середні, мінімальні та максимальні значення. Унаслідок цього результати експериментів стали придатними для коректного порівняння моделей між собою.

У межах дослідження окремо передбачено аналіз не лише продуктивності, а й енергоефективності. Для цього поряд із часовими показниками фіксувалися температура пристрою та параметри енергоспоживання. Такий підхід забезпечив можливість оцінити, наскільки досягнута продуктивність відповідає витраченим ресурсам, і сформував основу для подальшого виявлення більш ефективних режимів роботи моделей [5, 17, 27].

Таким чином, організація експериментів забезпечила однакові умови виконання моделей, узгоджений порядок збору показників та коректну підготовку результатів до подальшого аналізу. Реалізована методика стала основою для кількісного оцінювання продуктивності й енергоефективності моделей глибокого навчання на вбудованому пристрої та для перевірки працездатності розробленого модуля моніторингу.

3.5 Аналіз результатів оцінювання продуктивності та енергоефективності

У межах експериментального дослідження виконано оцінювання роботи моделей глибокого навчання на вбудованому обчислювальному пристрої за сукупністю часових, ресурсних, температурних та енергетичних показників. Аналіз результатів спрямовано на визначення відмінностей між досліджуваними моделями, виявлення більш ефективних режимів роботи та перевірку практичної придатності розробленого модуля моніторингу для комплексного оцінювання стану моделі.

Аналіз результатів виконано за кількома групами показників. До першої групи віднесено показники швидкодії, зокрема середній час виконання моделі, мінімальний і максимальний час, а також продуктивність. До другої групи включено характеристики використання апаратних ресурсів: середнє завантаження CPU, GPU та використання оперативної пам’яті. До третьої групи віднесено температурні та енергетичні показники, а саме середню температуру пристрою та середнє енергоспоживання під час виконання моделі. Такий підхід забезпечив не одновимірне, а комплексне оцінювання ефективності роботи моделей.

У таблиці 3.7 подано результати порівняння досліджуваних моделей за основними показниками продуктивності та енергоефективності.

Таблиця 3.7 – Порівняння результатів оцінювання моделей на вбудованому пристрої

Модель	Середній час виконання, с	Продуктивність, кадр/с	Середнє завантаження CPU, %	Середнє завантаження GPU, %	Середня температура, °C	Середнє енергоспоживання, Вт
MobileNetV2	0,032	31,25	24,8	46,2	55,4	4,1
ResNet18	0,058	17,24	31,6	62,8	60,7	5,3
ShuffleNet V2	0,028	35,71	22,9	41,5	53,8	3,9
EfficientNet-B0	0,049	20,41	29,4	57,1	58,9	4,8

Найменший середній час виконання зафіксовано для моделі ShuffleNet V2, яка також продемонструвала найвищу продуктивність і найнижче середнє

енергоспоживання. Модель MobileNetV2 забезпечила близькі результати та підтвердила високу придатність до використання на вбудованій платформі. Моделі ResNet18 та EfficientNet-B0 виявилися більш ресурсомісткими, що проявилося у збільшенні часу виконання, зростанні температури та підвищенні енергоспоживання.

Отже, за сукупністю показників найкращий баланс між швидкодією та енергоефективністю продемонстрували ShuffleNet V2 і MobileNetV2.

У процесі аналізу також визначено, що використання апаратних ресурсів має безпосередній вплив на характер роботи моделі. Якщо під час виконання фіксується високий рівень завантаження GPU за помірного використання CPU, це свідчить про орієнтацію обчислень на графічний прискорювач. Якщо ж навантаження на CPU залишається високим, це може вказувати на наявність додаткових накладних витрат, пов’язаних із підготовкою даних, передаванням тензорів або недостатньою оптимізацією виконання. Таким чином, результати моніторингу дали змогу не лише порівняти моделі, а й інтерпретувати причини відмінностей між ними.

Окрему увагу приділено температурному режиму та енергоспоживанню. У межах проведеного оцінювання підтверджено, що ці показники мають аналізуватися разом із продуктивністю. Зростання температури під час тривалого виконання моделі може бути ознакою надмірного навантаження на платформу, а підвищене енергоспоживання за відносно низької продуктивності є свідченням неефективного режиму роботи. Саме тому в аналітичному модулі використано інтегральний підхід, у межах якого результат оцінювання формується з урахуванням кількох груп показників одночасно.

На основі збережених результатів сформовано порівняльний графік індексу ефективності моделей. Цей показник використано для узагальненого зіставлення продуктивності та енергетичних витрат різних архітектур. Результати порівняння моделей наведено на рисунку 3.10.

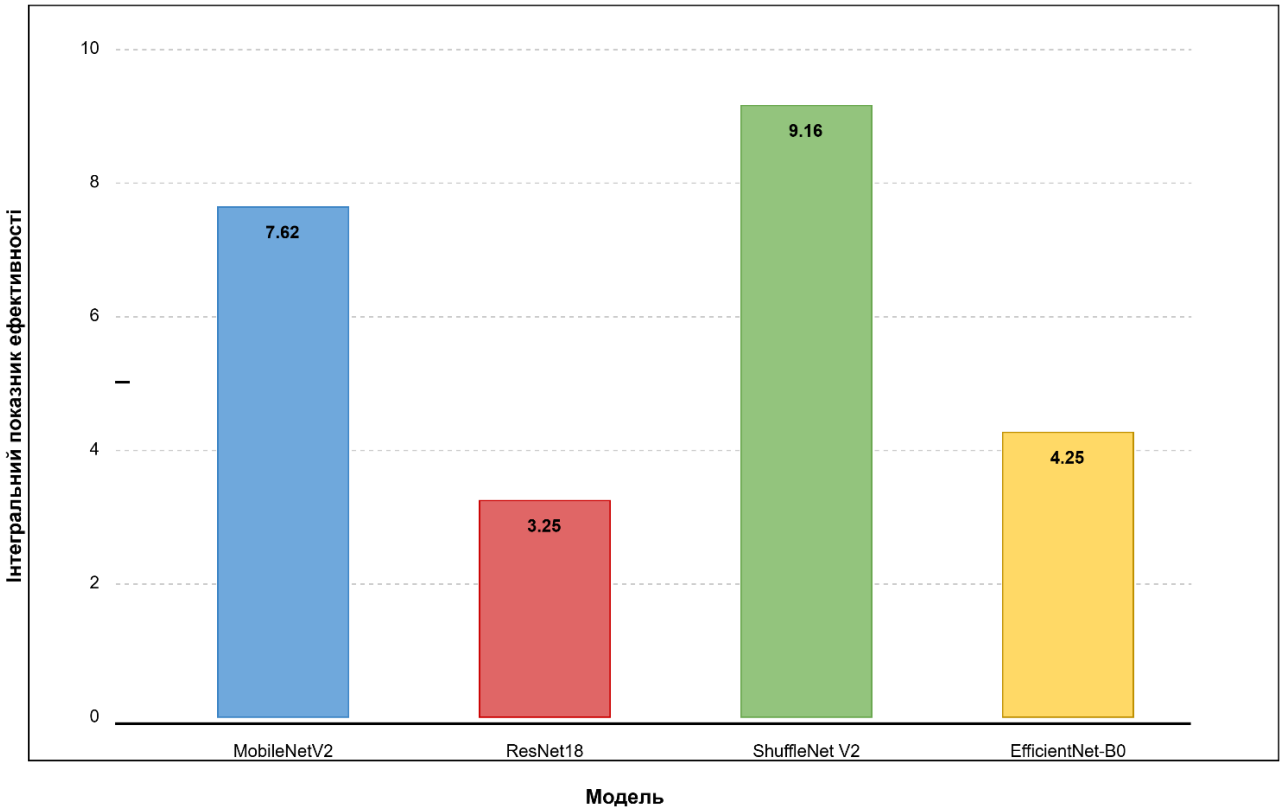

Рисунок 3.10 – Порівняння результатів оцінювання продуктивності та енергоефективності моделей

За результатами аналізу підтверджено працездатність розробленого модуля моніторингу. Реалізована система забезпечила одночасне одержання часових, системних і енергетичних показників, їх синхронізацію, узагальнення та формування підсумкових висновків. Унаслідок цього модуль став не лише засобом збору даних, а й інструментом підтримки прийняття рішень щодо вибору більш ефективної моделі або режиму її роботи на вбудованому пристрої.

Підвищення ефективності роботи моделей забезпечується шляхом використання полегшених архітектур, зменшення розміру вхідних даних, застосування оптимізованих режимів виконання та раціонального використання апаратних ресурсів. Для вбудованих платформ найбільш доцільним є використання моделей, що забезпечують збалансоване співвідношення між продуктивністю та енергоспоживанням, зокрема MobileNetV2 і ShuffleNet V2.

Додатково підвищення ефективності забезпечується за рахунок контролю температурного режиму пристрою, обмеження фонових навантажень та

використання засобів оптимізації виконання моделі. Це дозволяє зменшити ризик нестабільної роботи та забезпечити більш передбачувані результати під час тривалого використання системи.

Таким чином, аналіз результатів показав, що продуктивність і енергоефективність моделей глибокого навчання на вбудованій платформі повинні оцінюватися комплексно. Розроблений модуль моніторингу забезпечив таке оцінювання та створив основу для практичного порівняння моделей за сукупністю ключових характеристик.

ВИСНОВКИ

У кваліфікаційній роботі відповідно до поставлених завдань одержано такі результати:

1. Проаналізовано особливості застосування моделей глибокого навчання на вбудованих обчислювальних пристроях. Визначено, що ефективність їх використання залежить не лише від точності моделі, а й від часу виконання, навантаження на CPU і GPU, використання пам’яті, температурного режиму та енергоспоживання. Встановлено, що для embedded-платформ вирішальне значення має збалансованість продуктивності та ресурсної вартості виконання моделі.

2. Досліджено підходи до оцінювання продуктивності та енергоефективності моделей глибокого навчання. Визначено основні групи показників, які мають використовуватися для комплексного моніторингу: часові, ресурсні, температурні та енергетичні. Обґрунтовано, що коректне оцінювання моделей на вбудованих пристроях повинно виконуватися за сукупністю взаємопов’язаних метрик, а не за одним окремим показником.

3. Виконано аналіз сучасних засобів оптимізації та моніторингу моделей на edge- і embedded-платформах. Визначено можливості використання PyTorch, ONNX, TensorRT і системних засобів моніторингу платформи NVIDIA Jetson. Встановлено, що наявні інструменти здебільшого розв’язують окремі задачі, але не забезпечують цілісного аналітичного оцінювання продуктивності та енергоефективності в межах єдиного програмного модуля.

4. Сформовано вимоги до інтелектуального модуля моніторингу та розроблено його архітектуру. У структурі системи виділено модуль запуску моделі, модуль збору даних, модуль оброблення показників, аналітичний модуль і блок подання результатів. Розроблено структуру даних, яка охопила інформаційні об’єкти «Модель», «Запуск», «Системні дані», «Метрики» та «Рекомендації».

5. Розроблено алгоритми збору даних, аналізу метрик і формування рекомендацій. Запропоновано послідовність оцінювання стану моделі, яка включає отримання даних, перевірку їх коректності, обчислення узагальнених і похідних показників, порівняння з пороговими значеннями, визначення стану моделі та

формування рекомендацій щодо підвищення ефективності її роботи.

6. Реалізовано програмний прототип інтелектуального модуля моніторингу. Програмне рішення побудовано за модульним принципом із використанням Python, PyTorch і допоміжних засобів зчитування системних показників. Реалізовано функції завантаження моделей, вимірювання часу виконання, збору системних даних, обчислення узагальнених метрик, оцінювання стану моделі та збереження результатів.

7. Проведено експериментальне дослідження роботи модуля на вбудованій платформі. У межах експериментів оцінено роботу моделей MobileNetV2, ResNet18, ShuffleNet V2 та EfficientNet-B0. Порівняння результатів показало, що найбільш збалансовані показники продуктивності та енергоефективності продемонстрували легкі архітектури, зокрема ShuffleNet V2 і MobileNetV2. Підтверджено, що розроблений модуль забезпечує комплексне оцінювання стану моделі та дає змогу виявляти більш ефективні конфігурації її використання на embedded-платформі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Banbury C., Reddi V. J., Torelli P., Jeffries N., Kiraly C., Holleman J., Montino P., Kanter D., Ahmed S., Pau D. et al. MLPerf Tiny Benchmark. *Proceedings of Machine Learning and Systems*. 2021. Vol. 3. P. 452–468.

2. Banner R., Nahshan Y., Hoffer E., Soudry D. Post Training 4-bit Quantization of Convolutional Networks for Rapid-Deployment. *Advances in Neural Information Processing Systems*. 2019. Vol. 32.

3. Chen J., Ran X. Deep Learning with Edge Computing: A Review. *Proceedings of the IEEE*. 2019. Vol. 107, No. 8. P. 1655–1674. DOI: 10.1109/JPROC.2019.2921977.

4. Frankle J., Carbin M. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In: *International Conference on Learning Representations*. 2019.

5. Gupta U., Kim Y. G., Lee S., Tse J., Lee H.-H. S., Wei G.-Y., Brooks D., Wu C.-J. Chasing Carbon: The Elusive Environmental Footprint of Computing. In: *2021 IEEE International Symposium on High-Performance Computer Architecture*. 2021. P. 854–867.

6. Han S., Pool J., Tran J., Dally W. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In: *International Conference on Learning Representations*. 2016.

7. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016. P. 770–778.

8. Howard A. G., Zhu M., Chen B., Kalenichenko D., Wang W., Weyand T., Andreetto M., Adam H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint*. 2017. arXiv:1704.04861.

9. Huang G., Liu Z., Van Der Maaten L., Weinberger K. Q. Densely Connected Convolutional Networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017. P. 4700–4708.

10. Iandola F. N., Han S., Moskewicz M. W., Ashraf K., Dally W. J., Keutzer K. SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and <0.5 MB Model Size. *arXiv preprint*. 2016. arXiv:1602.07360.

11. Jacob B., Kligys S., Chen B., Zhu M., Tang M., Howard A., Adam H., Kalenichenko D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018. P. 2704–2713.

12. Jeong E., Kim J., Ha S. TensorRT-Based Framework and Optimization Methodology for Deep Learning Inference on Jetson Boards. *ACM Transactions on Embedded Computing Systems*. 2022. Vol. 21, No. 5. DOI: 10.1145/3508391.

13. Kochan O., Beshley M., Beshley H., Shkoropad Y., Ivanochko I., Seliuchenko N. SDN-based Internet of Video Things Platform Enabling Real-Time Edge/Cloud Video Analytics. In: *2023 17th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*. 2023.

14. Ma N., Zhang X., Zheng H.-T., Sun J. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In: *Proceedings of the European Conference on Computer Vision*. 2018. P. 116–131.

15. Mittal S. A Survey on Optimized Implementation of Deep Learning Models on the NVIDIA Jetson Platform. *Journal of Systems Architecture*. 2019. Vol. 97. P. 428–442.

16. Nvidia. *Jetson Nano Developer Kit*. NVIDIA Developer Documentation.

17. Nvidia. *Tegrastats Utility – Jetson Linux Developer Guide*. NVIDIA Documentation.

18. Nvidia. *TensorRT Developer Guide*. NVIDIA Documentation.

19. Oliveira F., Costa D. G., Assis F., Silva I. Internet of Intelligent Things: A Convergence of Embedded Systems, Edge Computing and Machine Learning. *Internet of Things*. 2024. Vol. 27. Art. 101153.

20. Open Neural Network Exchange. *ONNX Documentation*. ONNX Project Documentation.

21. PyTorch. *TorchVision Models Documentation*. PyTorch Documentation.

22. Qinghe Z., Tian X., Yang M., Su H. CLMIP: Cross-Layer Manifold Invariance Based Pruning Method of Deep Convolutional Neural Network for Real-Time Road Type Recognition. *Multidimensional Systems and Signal Processing*. 2021. Vol. 32. P. 1229–1248.

23. Reddi V. J., Cheng C., Kanter D., Mattson P., Schmuelling G., Wu C.-J., Anderson B., Breughe M., Charlebois M., Choupey P. et al. MLPerf Inference Benchmark. In: *ACM/IEEE 47th Annual International Symposium on Computer Architecture*. 2020. P. 446–459.

24. Sachenko A., Kochan V., Turchenko V., Tsahouridis K., Shendryk V., Dudnik O., Laopoulos T. Sensor Errors Prediction Using Neural Networks. In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN 2000)*. 2000. DOI: 10.1109/IJCNN.2000.860811.

25. Sandler M., Howard A., Zhu M., Zhmoginov A., Chen L.-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018. P. 4510–4520.

26. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In: *International Conference on Learning Representations*. 2015.

27. Swaminathan T. P., Silver C., Akilan T., Kumar J. Benchmarking Deep Learning Models on NVIDIA Jetson Nano for Real-Time Systems: An Empirical Investigation. *Procedia Computer Science*. 2025. Vol. 260. P. 906–913.

28. Szegedy C., Liu W., Jia Y., Sermanet P., Reed S., Anguelov D., Erhan D., Vanhoucke V., Rabinovich A. Going Deeper with Convolutions. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015. P. 1–9.

29. Tan M., Le Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In: *Proceedings of the 36th International Conference on Machine Learning*. 2019. P. 6105–6114.

30. Turchenko V., Kochan V., Sachenko A. Estimation of Computational Complexity of Sensor Accuracy Improvement Algorithm Based on Neural Networks. In: Dorffner G., Bischof H., Hornik K. (eds). *Artificial Neural Networks – ICANN 2001*. Lecture Notes in Computer Science. Vol. 2130. Berlin; Heidelberg: Springer, 2001. P. 743–748.

31. Wang P., Chen W., He X., Chen Q., Liu Q., Cheng J. Optimization-Based Post-Training Quantization with Bit-Split and Stitching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2023. Vol. 45, No. 2. P. 2119–2135.

32. Zhou Y., Yang K. Exploring TensorRT to Improve Real-Time Inference for Deep Learning. In: *2022 IEEE 24th International Conference on High Performance Computing and Communications*. 2022. P. 2011–2018. DOI: 10.1109/HPCC-DSS-SmartCity-DependSys57074.2022.00299.

33. Жураковський Б. Ю., Зенів І. О. *Технології Інтернету речей* : навч. посіб. Київ, 2021. 271 с.

34. Сініцин І. П., Дорошенко А. Ю., Смірнов В. Є., Яценко О. А. Дослідження продуктивності нейромереж YOLO для вбудованих систем і дронів. *Телекомунікаційні та інформаційні технології*. 2025. № 3. С. 177–186.

35. Хома Ю. В., Бенч А. Я. Порівняльний аналіз програмно-апаратного забезпечення алгоритмів глибокого навчання. *Комп'ютерні системи та мережі*. 2019. Вип. 1, № 1. С. 97–102.

36. Семенишин О.М. Інтелектуальний моніторинг продуктивності та енергоефективності моделей глибокого навчання на вбудованих обчислювальних пристроях. Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення (випуск 110): матеріали Міжнародної наукової інтернет-конференції, (м. Тернопіль, Україна, м. Ополе, Польща, 7-8 травня 2026 р.) / редкол. : О. Патряк та ін. ГО "Наукова спільнота", WSZIA w Opolu. Тернопіль : ФО-П Шпак В.Б. 2026. С. 67-69.

37. Комар М.П., Саченко А.О., Васильків Н.М та ін. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

38. Островерхов В. М., Біловус Л. І., Возьний К. З. та ін. Загальні методичні рекомендації з підготовки, оформлення, захисту та оцінювання кваліфікаційних робіт здобувачів вищої освіти першого (бакалаврського) і другого (магістерського) рівнів. Тернопіль: ЗУНУ, 2024. 83 с.

Додаток А

Код програмного забезпечення

Лістинг А.1 – Файл конфігурації системи (config.py)

```
from dataclasses import dataclass
from pathlib import Path


@dataclass
class AppConfig:
    model_name: str = "mobilenet_v2"
    input_width: int = 224
    input_height: int = 224
    batch_size: int = 1
    num_runs: int = 20
    sampling_interval: float = 0.5
    output_dir: str = "results"
    use_cuda: bool = True

    @property
    def output_path(self) -> Path:
        path = Path(self.output_dir)
        path.mkdir(parents=True, exist_ok=True)
        return path
```

Лістинг А.2 – Завантаження моделей глибокого навчання (model_loader.py)

```
from typing import Tuple

import torch
from torchvision import models


SUPPORTED_MODELS = {
    "mobilenet_v2": lambda: models.mobilenet_v2(weights="DEFAULT"),
    "resnet18": lambda: models.resnet18(weights="DEFAULT"),
    "shufflenet_v2_x1_0": lambda: models.shufflenet_v2_x1_0(weights="DEFAULT"),
    "efficientnet_b0": lambda: models.efficientnet_b0(weights="DEFAULT"),
}


def get_device(use_cuda: bool = True) -> torch.device:
    if use_cuda and torch.cuda.is_available():
        return torch.device("cuda")
    return torch.device("cpu")


def load_model(model_name: str, device: torch.device) -> torch.nn.Module:
    if model_name not in SUPPORTED_MODELS:
        raise ValueError(f"Непідтримувана модель: {model_name}")

    model = SUPPORTED_MODELS[model_name]()
    model.eval()
    model.to(device)
    return model


def create_input_tensor(
    width: int,
```

```
    height: int,
    batch_size: int,
    device: torch.device,
) -> torch.Tensor:
    return torch.randn(batch_size, 3, height, width, device=device)


def warmup_model(model: torch.nn.Module, input_tensor: torch.Tensor, steps: int =
3) -> None:
    with torch.no_grad():
        for _ in range(steps):
            _ = model(input_tensor)
```

Лістинг А.3 – Збір системних даних (system_monitor.py)

```
import re
import subprocess
import time
from typing import Dict, Optional

import psutil


def get_basic_system_data() -> Dict[str, float]:
    """Повертає базові системні показники, доступні на більшості платформ."""
    return {
        "cpu_load": psutil.cpu_percent(interval=None),
        "ram_usage_percent": psutil.virtual_memory().percent,
        "ram_used_mb": psutil.virtual_memory().used / (1024 * 1024),
        "temperature": get_cpu_temperature(),
    }


def get_cpu_temperature() -> float:
    """Повертає температуру CPU, якщо вона доступна."""
    try:
        temps = psutil.sensors_temperatures()
        for _, entries in temps.items():
            for entry in entries:
                if entry.current is not None:
                    return float(entry.current)
    except Exception:
        pass
    return 0.0


def parse_tegrastats_output(output: str) -> Dict[str, float]:
    """Парсить рядок tegrastats для Jetson-платформ."""
    result = {
        "gpu_load": 0.0,
        "temperature_gpu": 0.0,
        "power_value": 0.0,
    }

    gpu_match = re.search(r"GR3D_FREQ\s+(\d+)%", output)
    if gpu_match:
        result["gpu_load"] = float(gpu_match.group(1))

    temp_match = re.search(r"GPU@(\d+(?:\.\d+)?)C", output)
    if temp_match:
        result["temperature_gpu"] = float(temp_match.group(1))

    power_match = re.search(r"VDD_IN\s+(\d+)mW", output)
    if power_match:
        result["power_value"] = float(power_match.group(1)) / 1000.0
```

```
    return result


def get_jetson_data(timeout: float = 1.0) -> Dict[str, float]:
    """Зчитує дані tegrastats, якщо команда доступна."""
    try:
        process = subprocess.run(
            ["tegrastats", "--interval", "1000", "--logfile", "/dev/stdout"],
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            text=True,
            timeout=timeout,
        )
        line = process.stdout.strip().splitlines()[0] if process.stdout.strip()
else ""
        return parse_tegrastats_output(line)
    except Exception:
        return {"gpu_load": 0.0, "temperature_gpu": 0.0, "power_value": 0.0}


def get_system_data() -> Dict[str, float]:
    data = get_basic_system_data()
    data.update(get_jetson_data())
    data["timestamp"] = time.time()
    return data
```

Лістинг А.4 – Вимірювання часу виконання та продуктивності (performance_monitor.py)

```
import time
from typing import Dict

import torch


class PerformanceMonitor:
    def __init__(self) -> None:
        self.execution_times = []

    def measure_execution_time(self, model: torch.nn.Module, input_tensor:
torch.Tensor) -> float:
        if input_tensor.is_cuda:
            torch.cuda.synchronize()

        start_time = time.time()
        with torch.no_grad():
            _ = model(input_tensor)
        if input_tensor.is_cuda:
            torch.cuda.synchronize()
        end_time = time.time()

        execution_time = end_time - start_time
        self.execution_times.append(execution_time)
        return execution_time

    def calculate_performance(self, batch_size: int = 1) -> float:
        if not self.execution_times:
            return 0.0
        avg_time = sum(self.execution_times) / len(self.execution_times)
        if avg_time == 0:
            return 0.0
        return batch_size / avg_time
```

```
    def get_summary(self, batch_size: int = 1) -> Dict[str, float]:
        if not self.execution_times:
            return {
                "avg_execution_time": 0.0,
                "min_execution_time": 0.0,
                "max_execution_time": 0.0,
                "performance_value": 0.0,
            }

        return {
            "avg_execution_time": sum(self.execution_times) /
len(self.execution_times),
            "min_execution_time": min(self.execution_times),
            "max_execution_time": max(self.execution_times),
            "performance_value":
self.calculate_performance(batch_size=batch_size),
        }
```

Лістинг А.5 – Агрегування системних показників (metrics.py)

```
from typing import Dict, List


class MetricsCalculator:
    def __init__(self, system_records: List[Dict[str, float]]) -> None:
        self.system_records = system_records

    def _average(self, key: str) -> float:
        values = [record.get(key, 0.0) for record in self.system_records]
        return sum(values) / len(values) if values else 0.0

    def _maximum(self, key: str) -> float:
        values = [record.get(key, 0.0) for record in self.system_records]
        return max(values) if values else 0.0

    def calculate(self) -> Dict[str, float]:
        avg_temperature = self._average("temperature")
        avg_power = self._average("power_value")
        performance_value = self._average("performance_value")

        return {
            "avg_cpu_load": self._average("cpu_load"),
            "avg_gpu_load": self._average("gpu_load"),
            "avg_ram_usage": self._average("ram_usage_percent"),
            "avg_temperature": avg_temperature,
            "max_temperature": self._maximum("temperature"),
            "avg_power_value": avg_power,
            "efficiency_index": self.calculate_efficiency_index(performance_value,
avg_power),
        }

    @staticmethod
    def calculate_efficiency_index(performance_value: float, avg_power_value:
float) -> float:
        if avg_power_value <= 0:
            return 0.0
        return performance_value / avg_power_value
```

Лістинг А.6 – Оцінювання стану моделі та формування рекомендацій (analyzer.py)

```
from typing import Dict


class StateAnalyzer:
```

```
    def __init__(
        self,
        max_temperature: float = 75.0,
        max_cpu_load: float = 90.0,
        max_gpu_load: float = 95.0,
        min_efficiency: float = 0.5,
    ) -> None:
        self.max_temperature = max_temperature
        self.max_cpu_load = max_cpu_load
        self.max_gpu_load = max_gpu_load
        self.min_efficiency = min_efficiency

    def evaluate_state(self, metrics: Dict[str, float]) -> Dict[str, str]:
        avg_temperature = metrics.get("avg_temperature", 0.0)
        avg_cpu_load = metrics.get("avg_cpu_load", 0.0)
        avg_gpu_load = metrics.get("avg_gpu_load", 0.0)
        efficiency_index = metrics.get("efficiency_index", 0.0)

        problems = []

        if avg_temperature > self.max_temperature:
            problems.append("перевищення температури")
        if avg_cpu_load > self.max_cpu_load:
            problems.append("надмірне завантаження CPU")
        if avg_gpu_load > self.max_gpu_load:
            problems.append("надмірне завантаження GPU")
        if efficiency_index < self.min_efficiency:
            problems.append("низька енергоефективність")

        if not problems:
            return {
                "state_name": "нормальний",
                "problem_type": "відсутня",
                "recommendation_text": "Модель працює стабільно, зміна
конфігурації не потрібна.",
            }

        if len(problems) == 1:
            return {
                "state_name": "допустимий",
                "problem_type": problems[0],
                "recommendation_text": self.generate_recommendation(problems),
            }

        return {
            "state_name": "критичний",
            "problem_type": ", ".join(problems),
            "recommendation_text": self.generate_recommendation(problems),
        }

    @staticmethod
    def generate_recommendation(problems) -> str:
        if "перевищення температури" in problems:
            return "Зменшити навантаження на пристрій або перейти до оптимізованої
моделі."
        if "надмірне завантаження CPU" in problems:
            return "Оптимізувати підготовку даних або використати легшу
архітектуру."
        if "надмірне завантаження GPU" in problems:
            return "Зменшити розмір вхідних даних або використати оптимізований
режим виконання."
        if "низька енергоефективність" in problems:
            return "Перейти до оптимізованого режиму виконання або застосувати
полегшену модель."
        return "Потрібне додаткове налаштування конфігурації виконання моделі."
```

Лістинг А.7 – Збереження результатів у файл (storage.py)

```
import csv
from pathlib import Path
from typing import Dict, Iterable


class ResultStorage:
    def __init__(self, output_dir: Path) -> None:
        self.output_dir = output_dir

    def save_records(self, filename: str, records: Iterable[Dict]) -> Path:
        path = self.output_dir / filename
        records = list(records)
        if not records:
            return path

        with open(path, "w", newline="", encoding="utf-8") as csvfile:
            writer = csv.DictWriter(csvfile, fieldnames=records[0].keys())
            writer.writeheader()
            writer.writerows(records)
        return path
```

Лістинг А.8 – Головний сценарій роботи системи (main.py)

```
import time
from typing import Dict, List

from analyzer import StateAnalyzer
from config import AppConfig
from metrics import MetricsCalculator
from model_loader import create_input_tensor, get_device, load_model, warmup_model
from performance_monitor import PerformanceMonitor
from storage import ResultStorage
from system_monitor import get_system_data


def run_monitoring(config: AppConfig) -> Dict:
    device = get_device(config.use_cuda)
    model = load_model(config.model_name, device)
    input_tensor = create_input_tensor(
        width=config.input_width,
        height=config.input_height,
        batch_size=config.batch_size,
        device=device,
    )

    warmup_model(model, input_tensor)

    performance_monitor = PerformanceMonitor()
    analyzer = StateAnalyzer()
    storage = ResultStorage(config.output_path)

    system_records: List[Dict] = []
    execution_records: List[Dict] = []

    for run_index in range(config.num_runs):
        system_data_before = get_system_data()
        execution_time = performance_monitor.measure_execution_time(model,
input_tensor)
        system_data_after = get_system_data()

        performance_value = 0.0
        if execution_time > 0:
            performance_value = config.batch_size / execution_time
```

```
        execution_record = {
            "run_index": run_index + 1,
            "execution_time": execution_time,
            "performance_value": performance_value,
            "timestamp": time.time(),
        }
        execution_records.append(execution_record)

        merged_record = {
            **system_data_before,
            **system_data_after,
            "performance_value": performance_value,
        }
        system_records.append(merged_record)

        time.sleep(config.sampling_interval)

    performance_summary =
performance_monitor.get_summary(batch_size=config.batch_size)
    metrics_summary = MetricsCalculator(system_records).calculate()
    metrics_summary["performance_value"] =
performance_summary["performance_value"]
    state_result = analyzer.evaluate_state(metrics_summary)

    storage.save_records("system_records.csv", system_records)
    storage.save_records("execution_records.csv", execution_records)
    storage.save_records("summary.csv", [{**performance_summary,
**metrics_summary, **state_result}])

    return {
        "performance_summary": performance_summary,
        "metrics_summary": metrics_summary,
        "state_result": state_result,
    }


if __name__ == "__main__":
    config = AppConfig()
    result = run_monitoring(config)

    print("Підсумкові показники продуктивності:")
    for key, value in result["performance_summary"].items():
        print(f"{key}: {value}")

    print("\nПідсумкові системні та енергетичні показники:")
    for key, value in result["metrics_summary"].items():
        print(f"{key}: {value}")

    print("\nОцінювання стану моделі:")
    for key, value in result["state_result"].items():
        print(f"{key}: {value}")
```

Додаток Б

Копія публікації

УДК 001 (063)

Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення (випуск 110): матеріали Міжнародної наукової інтернет-конференції, (м. Тернопіль, Україна, м. Ополе, Польща, 7-8 травня 2026 р.) / редкол. : О. Патряк та ін. ГО "Наукова спільнота", WSZIA w Opolu. Тернопіль : ФО-П Шпак В.Б. 2026. 195 с. – ISSN 2522-932X

Збірник доповідей підготовлено за матеріалами Міжнародної наукової інтернет-конференції (випуск 110) 7-8 травня 2026 р. на сайті www.konferenciaonline.org.ua

Наша адреса: Оргкомітет МНІК "Конференція онлайн"
а/с 797, м. Тернопіль 46005
тел. моб. 068 366 0 525
e-mail: inetkonf@ukr.net

URL Інтернет-конференції: http://www.konferenciaonline.org.ua/
ISSN 2522-932X

***Семенишин Олександр Мирославович**, бакалавр,*
Західноукраїнський національний університет, м. Тернопіль

ІНТЕЛЕКТУАЛЬНИЙ МОНІТОРИНГ ПРОДУКТИВНОСТІ ТА ЕНЕРГОЕФЕКТИВНОСТІ МОДЕЛЕЙ ГЛИБОКОГО НАВЧАННЯ НА ВБУДОВАНИХ ОБЧИСЛЮВАЛЬНИХ ПРИСТРОЯХ

Інтернет-адреса публікації на сайті:
http://www.konferenciaonline.org.ua/ua/article/id-2590/

Стрімке поширення моделей глибокого навчання в сучасних інформаційних системах зумовило їх активне впровадження не лише в хмарних середовищах, а й на вбудованих обчислювальних пристроях. Такі моделі застосовуються в системах комп'ютерного зору, безпілотних платформах, промисловому контролі, робототехніці, сенсорних мережах, системах Інтернету речей та інших прикладних рішеннях, де важливими є автономність, низька затримка та оброблення даних у реальному часі. Перенесення інтелектуальної обробки безпосередньо на край мережі дає змогу зменшити залежність від хмарної інфраструктури, знизити навантаження на канали передавання даних і підвищити оперативність прийняття рішень [1, 2].

У межах проєктування інтелектуального модуля моніторингу розроблено структурну схему збору даних та оброблення показників роботи моделі, яка забезпечила одержання узгодженої інформації про стан вбудованого пристрою та параметри виконання моделі глибокого навчання. Побудована схема дала змогу поєднати системні характеристики платформи з показниками швидкодії моделі в межах єдиного потоку даних, придатного для подальшого аналітичного оцінювання [3].

У структурі збору даних виділено дві основні групи показників. До першої групи віднесено системні дані, які охоплюють завантаження CPU, завантаження GPU, використання оперативної пам'яті, температурні показники та параметри енергоспоживання вбудованого пристрою. До другої групи віднесено показники роботи моделі, зокрема час виконання одного запуску, середній час оброблення серії вхідних даних і продуктивність. Такий поділ дав змогу розмежувати характеристики стану апаратної платформи та безпосередні результати виконання моделі, а також забезпечив їх подальше спільне аналізування [4].

Збір даних організовано за принципом періодичного зчитування показників під час виконання моделі. У процесі роботи системи модуль моніторингу фіксує системні параметри пристрою через визначені часові інтервали, а також реєструє часові характеристики запуску моделі. Для кожного запису сформовано часову мітку, що забезпечило синхронізацію між змінами стану обчислювальної платформи та відповідними етапами виконання моделі. Такий підхід усунув розрізненість показників і сформував основу для побудови впорядкованих часових рядів.

Після завершення етапу збору виконано первинне оброблення даних. На цьому етапі забезпечено очищення неповних або некоректних значень, узгодження формату показників, агрегування даних та обчислення основних статистичних характеристик. Для кожної групи даних визначено середні, мінімальні та максимальні значення, що дало змогу перейти від набору окремих вимірювань до узагальненого представлення стану системи в межах одного циклу моніторингу.

У межах підсистеми оброблення також сформовано похідні метрики, які характеризують ефективність роботи моделі більш повно. До таких метрик віднесено середню продуктивність, зміну температури впродовж виконання, співвідношення між досягнутою продуктивністю та енергоспоживанням, а також інтегральні показники ефективності.

На рисунку 1 відображено послідовність руху даних у межах розробленого модуля. Початковою ланкою визначено вбудований пристрій із запущеною моделлю глибокого навчання. Від нього одночасно надходять два потоки даних: системні дані та показники роботи моделі. Далі виконано синхронізацію цих потоків за часовими мітками, після чого здійснено первинне оброблення й обчислення похідних метрик. Підготовлені результати передано до аналітичного модуля, у якому формується узагальнена оцінка ефективності роботи моделі.

У результаті розроблення підсистеми збору даних забезпечено цілісне поєднання системних і часових показників у межах єдиної структури.

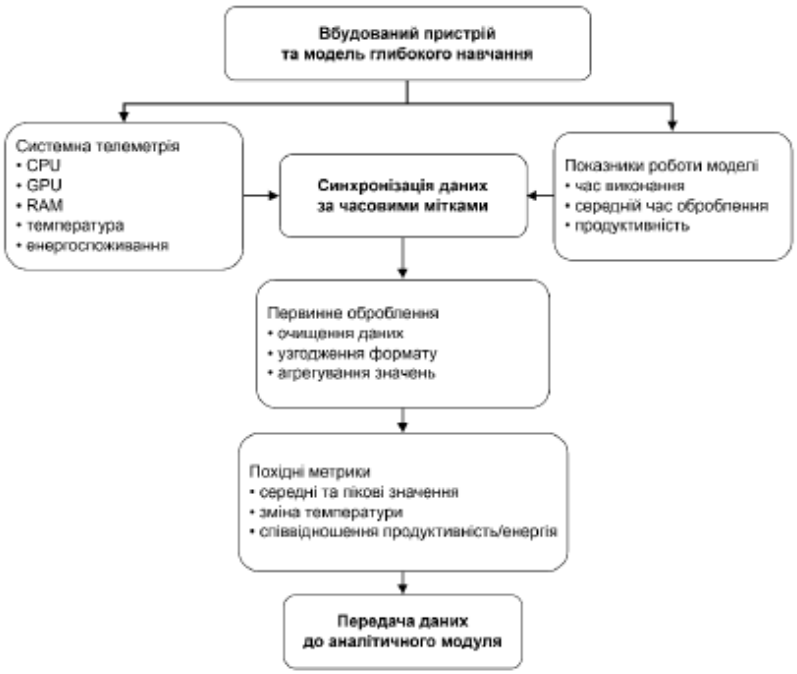

Рисунок 1 – Структурна схема збору даних та оброблення показників роботи моделі

Це дасть змогу підготувати дані до подальшого оцінювання продуктивності та енергоефективності моделі на вбудованій платформі, а також забезпечить інформаційну основу для формування рекомендацій щодо підвищення ефективності її роботи.

Література:

1. Oliveira F., Costa D. G., Assis F., Silva I. Internet of Intelligent Things: A Convergence of Embedded Systems, Edge Computing and Machine Learning. *Internet of Things*. 2024. Vol. 27. Art. 101153.
2. Жураковський Б. Ю., Зенів І. О. *Технології Інтернету речей*: навч. посіб. Київ, 2021. 271 с.
3. Banbury C., Reddi V. J., Torelli P., Jeffries N., Kiraly C., Holleman J., Montino P., Kanter D., Ahmed S., Pau D. et al. MLPerf Tiny Benchmark. *Proceedings of Machine Learning and Systems*. 2021. Vol. 3. P. 452-468.
4. Сініцин І. П., Дорошенко А. Ю., Смірнов В. Є., Яценко О. А. Дослідження продуктивності нейромереж YOLO для вбудованих систем і дронів. *Телекомунікаційні та інформаційні технології*. 2025. № 3. С. 177-186.