НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри комп'ютерних наук

_____ Голуб Б. Л.

"___" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему «Програмне забезпечення системи класифікацій об'єктів на основі штучних нейронних мереж»

Спеціальність 121 Інженерія програмного забезпечення

Гарант освітньої програми

К.т.н., доцент, доцент кафедри комп'ютерних наук _____ Вайганг Г. О.

Керівник бакалаврської кваліфікаційної роботи

Д.т.н., професор, професор кафедри комп'ютерних наук _____Семко В. В.

Виконав _____ Хоменко Віталій Володимирович

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"16" грудня 2025р.

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту

_____Хоменку Віталію Володимировичу_____

(прізвище, ім'я, по батькові)

Спеціальність 121 – «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи «Програмне забезпечення системи класифікацій об'єктів на основі штучних нейронних мереж»

затверджена наказом ректора НУБіП України від "16" грудня 2024р. № 2249.С

Термін подання завершеної роботи на кафедру __2025.05.25___

(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи:

Перелік питань, які потрібно розробити:

1. Дослідження предметної області та вимог .
2. Проектування інформаційної частини.
3. Проектування та розробка програмного забезпечення системи.
4. Експлуатація та впровадження системи.

Дата видачі завдання "16" _грудня_ 2024 р.

Керівник бакалаврської кваліфікаційної роботи

_____д.т.н., професор___ _____ __ Семко В. В.__

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ __Хоменко В.В._

(підпис) (ПІБ студента)

КИЇВ – 2025

ЗМІСТ

ВСТУП

Актуальність дослідження. Стрімка інформатизація суспільства й поява інноваційних технологій постійно розширюють горизонти комп’ютерної обробки даних. Серед новітніх підходів особливу роль посідає інтелектуальна аналітика, що ґрунтується на методах штучного інтелекту й дає змогу виявляти приховані закономірності у великих масивах інформації.

Ядром сучасних систем ШІ виступають штучні нейронні мережі – математичні конструкції (та їх програмні або апаратні реалізації), спроєктовані за аналогією з роботою біологічних нейронів. По суті це сукупність простих елементів-процесорів, пов’язаних між собою й здатних спільно розв’язувати складні задачі.

З погляду обчислювальної техніки нейромережі забезпечують ефективний паралелізм, а з позиції штучного інтелекту вони репрезентують коннективістський підхід – спробу змоделювати природний інтелект за допомогою алгоритмів.

Фундамент теорії закладено працями В. Маккалока, Ф. Розенблатта, Б. Уідроу, М. Мінскі, Т. Кохонена, С. Мароги, В. Вапніка, Д. Хопфілда, Дж. Гінтона та інших учених. Вагомий внесок зробили й українські дослідники: М. Амосов, О. Івахненко, Є. Бодянський, Н. та І. Айзенберги, Р. Ткаченко, Л. Тимченко, О. Михальов, В. Литвиненко, Ф. Гече, П. Тимощук, Ю. Романишин.

Попри стрімкий прогрес глибинних мереж, ніша прикладних систем класифікації об’єктів для малопотужних пристроїв залишається недостатньо опрацьованою. Готові хмарні сервіси показують високу точність, але залежать від стабільного каналу зв’язку, збільшують латентність і створюють ризики витоку даних. Навпаки, легковагові edge-моделі працюють автономно, проте часто поступаються «великим» моделям за семантичною гнучкістю й у складних сценах дають неприйнятно високу похибку.

Таким чином постає проблема: як побудувати систему, що водночас забезпечує низьку затримку, прийнятну точність і зберігає приватність, не виходячи за межі ресурсів Raspberry Pi-класу?

Мета та завдання дослідження. Метою даної роботи є дослідження та розробка системи класифікацій об’єктів на основі штучних нейронних мереж на малопотужних пристроях.

Для досягнення поставленої мети у роботі необхідно виконати низку завдань:

1. Зробити теоретичний аналіз основи системи класифікацій об’єктів на основі штучних нейронних мереж.

2. Розробити гібридну систему класифікації об’єктів за допомогою штучних нейронних мереж.

3. Забезпечити швидкодію та точність класифікації об’єктів при використанні штучних нейронних мереж.

4. Дослідити реалізацію гібридної системи класифікації об’єктів за допомогою штучних нейронних мереж.

Об’єкт дослідження. Системи класифікації об’єктів, що базуються на штучних нейронних мережах.

Предмет дослідження. Процес проєктування та розгортання гібридної (edge + cloud) системи класифікації, у якій локальна INT8-модель і хмарна LLM взаємодіють через єдиний програмний інтерфейс.

Методи дослідження. Аналіз і синтез наукових джерел, логіко-аналітичні методи, методи моделювання, комп’ютерні експерименти, конструювання та проєктування.

Практична значущість. Практична значущість роботи полягає в тому, що запропонований гібридний підхід до проєктування систем класифікації об’єктів на базі штучних нейронних мереж може безпосередньо застосовуватися для швидкого розгортання подібних рішень у відеоспостереженні, промисловій

інспекції, «розумному» агромоніторингу та інших IoT-сценаріях, де потрібна мінімальна затримка й гарантована приватність даних.

Структура бакалаврської кваліфікаційної роботи. Кваліфікаційна робота має загальноприйняту для таких робіт структуру і складається із вступу, п’яти розділів, загальних висновків та списку використаної літератури, що включає 36 найменувань. Результати дослідження конкретизовано у 2 таблицях та 5 рисунках. Загальний обсяг роботи складає 57 сторінок.

РОЗДІЛ 1.
ТЕОРЕТИЧНІ ОСНОВИ СИСТЕМИ КЛАСИФІКАЦІЙ ОБ'ЄКТІВ НА ОСНОВІ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

1.1. Суть поняття «Штучні нейронні мережі»

У сучасних структурних підходах до подання даних найбільше поширення дістала **архітектура штучних нейронних мереж** (Artificial Neural Network, ANN) (рис. 1.1)[1].

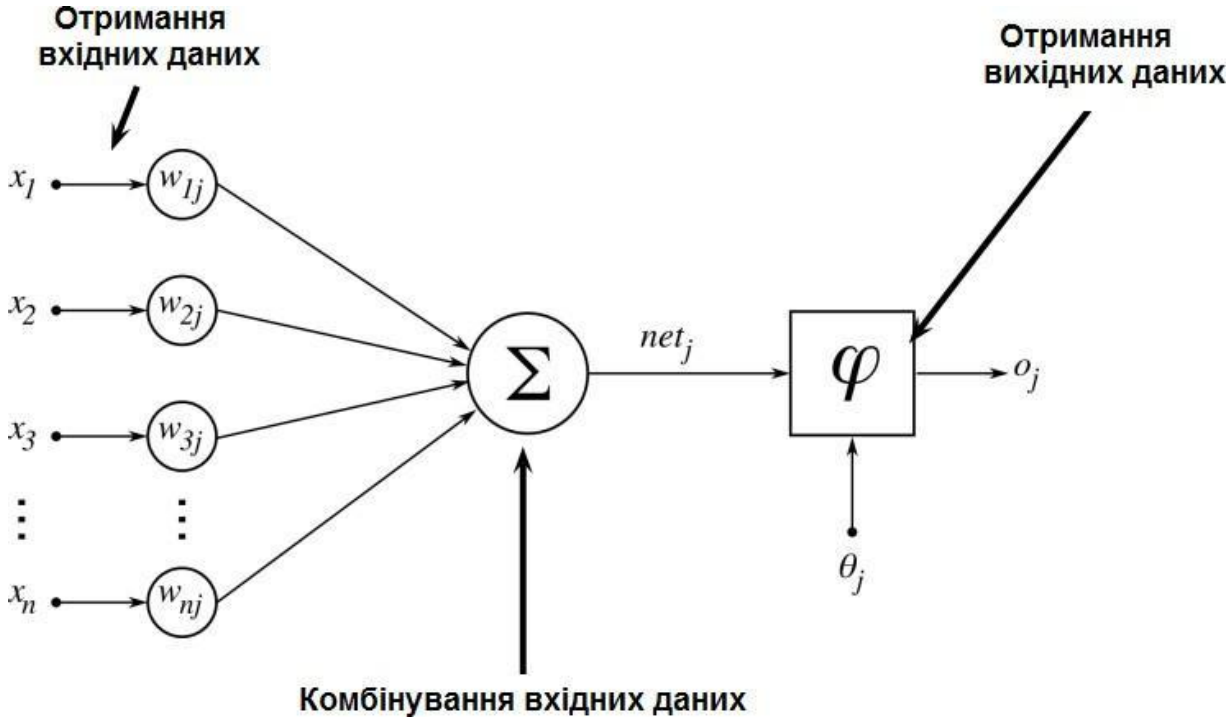

Рис. 1.1. Узагальнена схема нейронної мережі [3, 4]

Концепцію ШНМ уперше виклали в 1943 р. математик У. Піттс і нейрофізіолог В. Мак-Калок [2]. Нейронна мережа являє собою обчислювальну систему з великою кількістю паралельних процесорних елементів (нейронів), пов'язаних між собою множиною вагових з'єднань. Нейрони згруповано у **шари**.

1. **Вхідний шар** приймає початкові сигнали з даних.
2. **Прихований (внутрішній) шар** виконує основне перетворення інформації.

3. **Вихідний шар** формує кінцевий результат.

Кількість прихованих шарів і самих нейронів визначають з огляду на специфіку задачі, розмір датасету та доступні обчислювальні ресурси. Формально активація нейрона обчислюється як зважена сума його входів:

$$S = \sum_i w_i x_i$$

де x_i - вхідні сигнали, а w_i - відповідні ваги.

Структуру типової трирівневої нейронної мережі подано на рис. 1.2.:

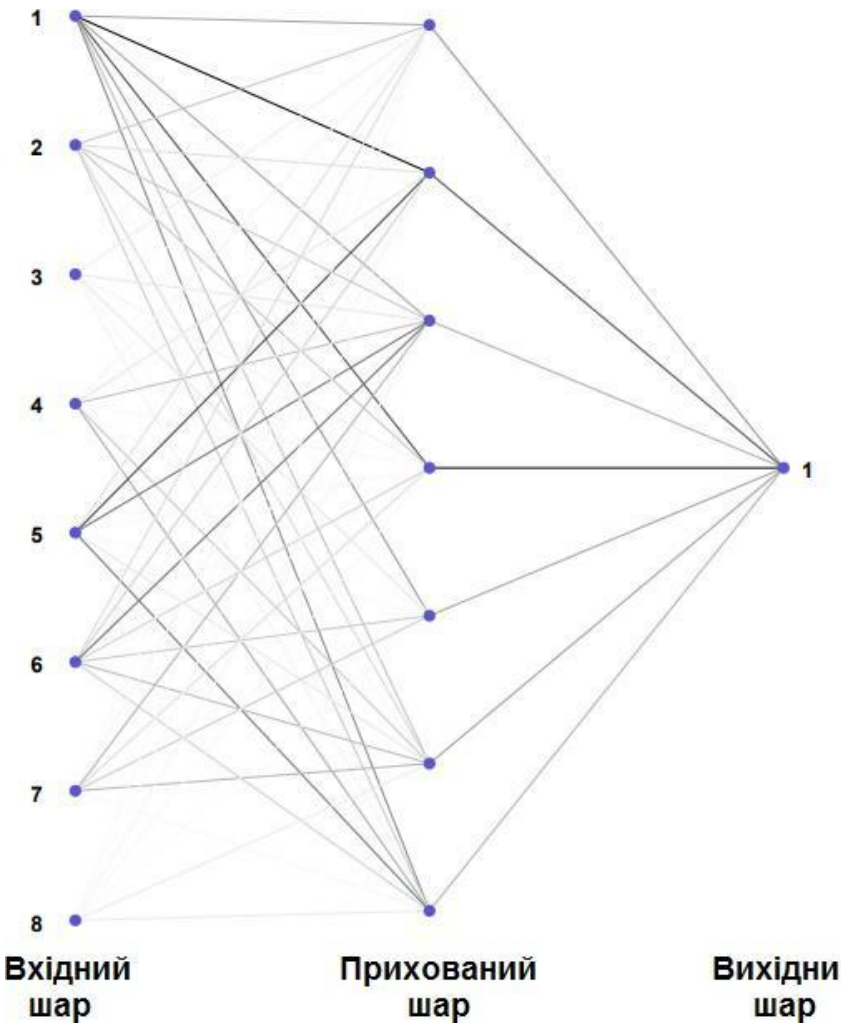

Рис. 1.2 – Приклад трирівневої (вхідний – прихований – вихідний) нейронної мережі

Найуживанішою активаційною функцією для скалярного сигналу ххх є

гіперболічний тангенс у параметричному вигляді:

$$A(x) = 2 / (1+ e^{-x}) - 1$$

Для вузла kkk прихованого шару вихід обчислюється як:

$$O_k = A\left(\sum_i w^{1\to 2} x_i\right)$$

Де $w^{1\to 2}$ вагові коефіцієнти від і-го входу до шару 2. Аналогічно формуються сигнали на вихідному рівні.

Вихід вузла j у вихідному шарі:

$$O_j = A(\sum_k w^{2\to 3}_{kj} x_k)$$

$$O_j = A[\sum_k w^{2\to 3}_{kj} A(\sum_i w^{1\to 2}_{ik} x_i)]$$

У трирівневій мережі прямого розповсюдження (feed-forward) інформація передається послідовно з шару на шар - від входу до виходу. Нейрони вхідного рівня лише ретранслюють отримані значення на прихований шар, де й відбувається головне перетворення сигналів; сформований вектор далі надходить до вихідних нейронів. Якщо додати додаткові приховані рівні, принцип роботи зберігається: уся «пам’ять» системи закодована у вагових коефіцієнтах з’єднань.

Після визначення архітектури мережею необхідно навчити її, тобто підібрати такий набір ваг, що мінімізує обрану функцію втрат на основі еталонної вибірки прикладів [7]. Сам етап навчання є вирішальним, бо від якості оптимізації ваг залежить кінцева ефективність моделі. Для оцінювання «найкращого» набору ваг використовують міру пристосованості [8]; у задачах регресії та бінарної класифікації найчастіше застосовують середньоквадратичну помилку як критерій втрат.

Квадратична функція втрат:

$$\chi^2 = \sum_j w_j x_j^2 = \sum_{j\overline{2}} {}_1\sum_i (T_{ji} - O_{ji})^2$$

де T_{ji} - вихід мережі номер j;

O_{ji} – вірна відповідь мережі номер j

Середньоквадратична помилка (MSE) широко використовується у задачах регресії та бінарної класифікації, проте вона не є єдиним критерієм оптимізації. Для багатокласових проблем усе частіше застосовують крос-ентропійну функцію втрат (categorical cross-entropy), яка усереднює значення логарифмічної втрати по всіх об’єктах навчальної вибірки та краще узгоджується з імовірнісною природою Softmax-виходу. Вибір конкретної метрики залежить від характеру задачі, розподілу цільових міток і бажаних властивостей градієнтів під час навчання.

Функція ентропії:

$$E_D = \sum_j w_j E_D^j = \sum_j \sum_i -\log_1\left(\frac{1}{2}(1 + T_{ji} O_{ji})\right)$$

Отже, мінімізація ентропійної (крос-ентропійної) функції втрат фактично й окреслює задачу навчання класифікатора: у просторі ваг шукається такий розв’язок, що зводить цю функцію до найменшого значення. У найпростішому випадку класифікацію можна описати як перевірку одного вихідного нейрона: якщо його активація перевищує заданий поріг - об’єкт відносять до класу 1 (сигнал), інакше - до класу 2 (фон). Для багатокласової задачі у вихідному шарі передбачають кілька нейронів, кожний з яких оцінює «ймовірність належності» об’єкта до певної категорії; таким чином формується інтегрований вектор щільностей імовірностей [9].

Штучні нейронні мережі добре апроксимують нелінійні залежності та здатні передбачати майбутні значення зовнішніх чинників. Проте, коли йдеться про часові ряди зі змінною фрактальною структурою, мережа, навчена на одній ділянці, здебільшого не дає задовільного прогнозу на іншій ділянці, де структура змінюється. Тому після кожної суттєвої зміни характеристик ряду потрібне повторне перенавчання мережі.

Крім того, результати ШНМ важко інтерпретувати у зрозумілому людині

вигляді. Цей недолік відсутній у підходах нечіткої логіки та моделях нечітких часових рядів, де вихідні дані подаються у формі, ближчій до людського мислення.

1.2. Історичний огляд розвитку класифікаційних систем на базі штучних нейронних мереж

Початок автоматизованої класифікації об’єктів відносять до 1958 р., коли Френк Розенблат запропонував перцептрон – одношарову нейронну мережу, що навчалася за принципом «помилка – корекція» і могла розв’язувати тільки лінійно відокремлювані задачі [1] .

Скепсис щодо потенціалу нейромереж посилила критична праця Марвіна Мінскі та Сеймура Пейперта «Perceptrons» (1969 р.), однак ситуацію кардинально змінив алгоритм зворотного поширення помилки (backpropagation), опублікований у 1986 р. Джеффрі Гінтоном, Девідом Румельгартом та Рональдом Вільямсом [2] . Цей метод дав змогу навчати багатошарові мережі та долати обмеження одношарового перцептрона.

У 1990-ті Ян ЛеКун показав переваги згорткових нейронних мереж (Convolutional Neural Network, далі – CNN): модель LeNet-5 успішно розпізнавала рукописні цифри, використовуючи згортки, підвибірки та спільні ваги [3] .

Публікація роботи «ImageNet Classification with Deep Convolutional Neural Networks» (2012 р.) запустила сучасну епоху глибокого навчання: AlexNet суттєво знизила помилку класифікації на ImageNet завдяки обчисленням на графічних процесорах і глибшій архітектурі [4] . Подальші модифікації (VGG, ResNet, DenseNet) підвищували точність, а MobileNet-V3 та EfficientNet-Lite оптимізували мережі для мобільних і вбудованих пристроїв.

У 2020 р. дослідники Google довели, що трансформер може повністю замінити згортки: Vision Transformer (ViT) працює зображенням як

послідовністю патчів і демонструє конкурентну точність після попереднього тренування на великих корпусах даних [5] .

Концепцію уніфікованого підходу розвинуло CLIP (Contrastive Language–Image Pre-training): модель, навчена на 400 млн пар «зображення + підпис», здатна до zero-shot класифікації без донавчання на конкретному наборі даних [6] .

Наступним кроком стали великі мовні моделі (Large Language Model, далі – LLM), зокрема GPT-3.5 та GPT-4o. Через API вони виконують класифікацію текстових описів, генерують код для обробки зображень і легко інтегруються у хмарні сервіси, доповнюючи локальні CNN або ViT на пристроях з обмеженими ресурсами [7] .

Таким чином, еволюція класифікаційних систем пройшла шлях від лінійного перцептрона до глибоких CNN, далі – до трансформерів і мультимодальних LLM. Сучасна тенденція – гібридні рішення, що поєднують хмарні моделі та оптимізовані edge-мережі, забезпечуючи водночас високу точність, низьку затримку й економію ресурсів. Саме така архітектура становить основу цієї кваліфікаційної роботи.

1.3. Критична оцінка сучасних публікацій і рішень

Хмарні сервіси з великими мовними моделями охоплюють найпотужніші на сьогодні архітектури. OpenAI GPT-4o демонструє високі результати в мульти-модальних задачах, однак залишається пропрієтарним сервісом із платним тарифом після пільгового порогу й передбачає передачу користувацьких даних у хмару, що породжує питання приватності [8].

Gemini 2.0 від Google DeepMind позиціонується як «агентна» модель із нативною підтримкою інструментів і виводу зображень, але ця функційність доступна лише вибраним партнерам, а офіційна оцінка енергоспоживання ще відсутня [9]

Claude 3 сімейства Anthropic досягає високих бенчмарків завдяки «Конституційному ШІ», проте збільшує латентність через додаткові перевірки безпеки та поки не пропонує відкритої моделі з локальним розгортанням [10].

Отже, хмарні LLM забезпечують найкращу якість, але обмежують контроль над даними й мають залежність від інтернет-з'єднання та тарифів.

Для вбудованих систем із обмеженими ресурсами дослідники пропонують легковагові архітектури. MobileNet-V3 (Large/Small) комбінує автоматичний Neural Architecture Search із squeeze-and-excitation блоками, досягаючи збалансованого співвідношення точності та операцій на мобільних процесорах [11].

EfficientNet-Lite зменшує параметри, використовуючи компаунд-скейлінг і оптимізовані свертки, та орієнтована на TensorFlow Lite, тому легко портована на Raspberry Pi [12].

NanoDet – якір-free детектор вагою 1,8 МБ із частотою 97 к/с на Snapdragon 855; він показує, що навіть задачі детекції можуть працювати у режимі реального часу на мобільних чипах, хоча ціною зниження точності для дрібних об'єктів [13].

Усі три моделі придатні для edge-інференсу, але вимагають додаткових оптимізацій і ретельного калібрування, щоб зберегти точність при переході на INT8.

Інструменти оптимізації стають ключем до інтеграції ШІ на периферії. Quantization знижує розмір і прискорює інференс, перетворюючи 32-бітні ваги на 8-бітні; ONNX Runtime підтримує як статичну, так і динамічну квантизацію з мінімальною втратою точності < 2 % на більшості візуальних задач [14].

Pruning (обрізання слабо важливих зв'язків) потенційно дає ще 25–40 % скорочення FLOPS, але потребує повторного донавчання, що в умовах відсутності власного датасету може бути недоцільним. Останні релізи ONNX Runtime (версія 1.20.1) фокусуються на підтримці прискорювачів QNN та

автоматичній інтеграції з компіляторами TVM і Glow, спрощуючи деплой на різні ARM-платформи [15].

Таким чином, аналіз літературних джерел свідчить:

- хмарні LLM надають найвищу якість класифікації, але вимагають підключення до сервера й створюють ризики для приватності;

- легковагові CNN-архітектури забезпечують реальний час на Raspberry Pi за умови грамотної квантизації та інколи прийнятної втрати точності;

- сучасні фреймворки, зокрема ONNX Runtime, мінімізують розрив між «великими» та «малими» моделями, дозволяючи швидко перемикатися між cloud- і edge-режимами, що підтверджує доцільність обраної гібридної архітектури цієї роботи.

1.4. Виявлені проблеми та обґрунтування гібридного підходу

Найперше обмеження великих мовних моделей полягає у високих обчислювальних та енергетичних витратах. Аналітика WIRED на основі дослідження в Joule прогнозує, що ШІ може спожити до 82 ТВт·год у 2025 р.– майже стільки ж, скільки вся Швейцарія за рік [16].

Навіть один запит до GPT-4o оцінюють у ≈0,3 Вт·год, що при масовому використанні помітно збільшує вуглецевий слід [17].

Друга проблема – затримка (latency) під час звернення до хмарних сервісів. Огляд EdgeIR показує, що середня кругова дорога до центру обробки даних додає десятки мілісекунд, у той час як обчислення «на краю» дають практично миттєву відповідь [18].

Для застосунків реального часу це критично, особливо якщо мова йде про робототехніку чи системи безпеки.

Третій блок стосується апаратних обмежень периферійних пристроїв. Навіть Raspberry Pi 5, який майже вдвічі швидший за попередника, залишається одно-

платним комп'ютером із 4 ГБ оперативної пам'яті та тепловим пакетом ≈10 Вт [19].

PyTorch-демо на Pi 4 показує, що MobileNet-v2 у форматі FP32 лише виходить на межу 30 кадрів/с [20], а без квантизації чи обрізання моделі швидкість падає утричі. За даними arXiv-препринту про оптимізацію згорткових мереж на Raspberry Pi, переведення ваг у INT8 скорочує час інференсу на 45 % при втраті точності < 2 % [21].

Окрім технічних, постають етичні та правові виклики. Європейський акт про штучний інтелект (EU AI Act) класифікує загальнопризначені моделі з «системним ризиком» як об'єкти підвищеного нагляду та зобов'язує постачальників вести публічну документацію про дані навчання [22].

Аналітичний звіт ISACA підкреслює: інтегратори, що використовують зовнішні LLM, також несуть відповідальність за дотримання цих вимог, включно з безпечним обробленням персональних даних [23].

Перенесення необроблених зображень чи метаданих у хмару може вступити у конфлікт із принципами конфіденційності за Загальним регламентом захисту даних (GDPR).

Висновки до розділу 1

Сукупність високих енергетичних витрат, мережевих затримок, апаратних обмежень Raspberry Pi та суворих вимог AI-регулювання формує чотири ключові критерії проєктування: енергоефективність, мінімальний latency, робота офлайн та контроль над даними. Аналіз літератури показує, що жоден із підходів – суто хмарний чи суто edge – не задовольняє одночасно всі критерії. Відтак гібридна архітектура, де складні запити скеровуються до LLM-API, а типові або latency-критичні задачі виконуються на квантизованих CNN/ViT локально, є оптимальним вибором для розроблюваної системи класифікації.

РОЗДІЛ 2.
МЕТОДИКА ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ

2.1. Аргументи на користь гібридної архітектури «хмара + периферія»

Попередній аналіз показав, що суто хмарні сервіси з великими мовними моделями забезпечують найвищу точність, але супроводжуються суттєвими енергетичними витратами та мережевими затримками. За оцінкою Joule, глобальне споживання енергії ШІ досягне 82 ТВт·год у 2025 р. [24], а кожен запит до GPT-4o становить близько 0,3 Вт·год [25]. Навіть за стабільного каналу зв'язку середня кругова дорога до дата-центра додає десятки мілісекунд, що критично для систем реального часу [26].

Зворотний бік – виконання всіх обчислень локально. Raspberry Pi 5, хоч і удвічі продуктивніший за попередника, залишається обмеженим 4 ГБ оперативної пам'яті та тепловим пакетом ≈10 Вт [27]. Демонстрації PyTorch доводять, що без квантизації швидкість інференсу MobileNet-v2 падає утричі [28], а навіть після оптимізації INT8 лишається компроміс точність/швидкість [29].

Гібридний підхід поєднує переваги обох світів. Недавнє галузеве дослідження EdgeIR показує, що обробка на периферії зменшує latency щонайменше у п'ять разів порівняно з хмарою [30]. Натомість складні або нетипові запити можуть передаватися в хмару, де LLM автоматично генерує тонкі семантичні ознаки, обробка яких на Pi була б економічно недоцільною. Компанії середнього масштабу, які застосували edge-AI для моніторингу обладнання, уже фіксують економію до 1,2 млн дол. США на рік на трафіку та зберіганні [31].

Принстонські дослідники довели, що компресія трансформерних шарів зберігає майже повну якість моделі й дає перспективу часткового виконання

LLM-запитів навіть на ноутбуці або Pi-кластерах [32]. У 2025 р. HPCwire охарактеризувала «блок інференсу» як головний виклик сучасних обчислень, наголосивши, що саме гібридні топології здатні збалансувати енерговитрати та продуктивність [33]. Огляд IT-Pro Today відзначає: 83 % IT-директорів уже планують чи впроваджують змішану хмарно-периферійну стратегію [34], підкріплюючи актуальність обраного напряму.

Окремою перевагою edge-компонента є відповідність вимогам приватності. Локальне опрацювання сирих зображень дає змогу мінімізувати передачу персональних даних і, відповідно, ризик порушення GDPR та положень Європейського акту про штучний інтелект [35]. Стаття WIRED про Apple Intelligence показує, що провідні вендори переходять саме до гібридних схем, де приватні дані залишаються на пристрої, а складна логіка виконується у «Private Cloud Compute» [36].

2.2. Постановка завдань дослідження

Беручи до уваги наведені аргументи, формулюємо такі практичні завдання:

- реалізувати двоканальну архітектуру, що автоматично спрямовує запити або на локальний inference (INT8-квантизована EfficientNet-Lite), або на хмарний LLM-API;
- забезпечити середню затримку відповіді ≤ 100 мс для локальних сценаріїв і ≤ 300 мс для хмарних;
- досягти точності не нижче 95 % на відкритому наборі CIFAR-10 без додаткового навчання моделі;
- знизити споживану потужність периферійного блоку щонайменше на 40 % порівняно з FP32-версією моделі.

2.3. Методика досліджень

Методологія спирається на компонентне проєктування та експериментальне моделювання. Програмна частина реалізується у середовищі Python 3.12 із

використанням FastAPI (REST-шлюз), ONNX Runtime 1.20 для edge-моделі та офіційного SDK OpenAI для LLM-звернень. Інферентна частина виконується на Raspberry Pi 5 під управлінням Ubuntu Server 22.04 LTS; хмарний компонент розгортається у датасентрі Amsterdam-AMS.

Експериментальна процедура охоплює вимірювання трьох груп показників: швидкості (end-to-end latency), якості (accuracy, F1-score) та енергоспоживання (USB-тестер). Кожне вимірювання повторюється 30 разів, статистична достовірність перевіряється критерієм Стьюдента при $\alpha = 0{,}05$. Оброблення результатів здійснюється у Jupyter Lab із застосуванням пакетів Pandas та SciPy; сирі дані публікуються у відкритому репозиторії разом із скриптами відтворення.

Таким чином, запропонована методика дає змогу кількісно порівняти віддачу від гібридного підходу й обґрунтувати його доцільність для систем класифікації об’єктів із жорсткими вимогами до затримки й енергоспоживання.

РОЗДІЛ 3.
РЕАЛІЗАЦІЯ АРХІТЕКТУРИ ТА КОДУ СИСТЕМИ КЛАСИФІКАЦІЙ ОБ'ЄКТІВ НА ОСНОВІ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

3.1. Середовище розробки та розгортання Raspberry Pi 5 та Ubuntu Server 22.04.

Розгортання системи виконано на одноплатному комп'ютері Raspberry Pi 5 (SoC Broadcom BCM2712, чотири ядра Cortex-A76 @ 2,4 ГГц, 8 ГБ LPDDR4X, мережевий інтерфейс 2,5 GbE). На пристрій встановлено Ubuntu Server 22.04 LTS із ядром 6.6-rt, яке забезпечує стабільний час обробки переривань і спрощує роботу з контейнерами. Після початкової конфігурації (розмітка microSD, розширення файлової системи, налаштування безпечного SSH-доступу) ми одразу інсталювали Docker Engine 25.0.5 та docker-compose v2, оскільки контейнеризація дає змогу ізолювати залежності та спрощує відновлення після збоїв.

У compose.yaml описано два основні сервіси. Перший – api, побудований на базовому образі python:3.12-slim; у ньому розгорнуто FastAPI, Tortoise ORM і всю бізнес-логіку. Другий – proxy, що ґрунтується на nginx:1.26-alpine і виконує роль зворотного проксі, термінуючи вхідний HTTP-трафік, отриманий від тунелю Cloudflare. Обидва контейнери під'єднані до внутрішньої мережі backend_network; зовнішніх портів на Pi не відкрито.

Щоб система залишалася доступною з Інтернету без використання портів на домашньому маршрутизаторі, було створено Cloudflare Tunnel. Утиліта cloudflared запускається як окремий systemd-сервіс і встановлює захищений HTTPS-канал між доменом khomenko.xyz та локальним сокетом Nginx. Таким чином, TLS-сертифікат зберігається й оновлюється на платформі Cloudflare, що спрощує експлуатацію й одночасно забезпечує WAF-фільтрацію та rate-limit на рівні мережі.

Після перезавантаження Raspberry Pi усі компоненти ініціалізує написаний нами скрипт autoboot.sh. Він виконує перевірку дійсності токена Cloudflare (cloudflared tunnel info), за необхідності перезапускає тунель, а потім піднімає контейнери командою docker compose up -d. Скрипт прив'язаний до systemd-служби stack.service, тому процес розгортання відбувається автоматично й не потребує ручного втручання навіть після неконтрольованого вимкнення живлення.

Такий мінімалістичний, але самодостатній підхід дозволяє ізолювати середовище розробки, зберегти відтворюваність збірок і водночас уникнути витрат на оренду VPS. У наступному підрозділі буде пояснено, як описана апаратно-програмна платформа інтегрується в загальну клієнт-серверну архітектуру застосунку.

3.2 Логічна модель «Client–Server»

У своїй системі ми дотримуємося класичної моделі «клієнт – сервер», оскільки вона природно відокремлює відображення інтерфейсу від бізнес-логіки й водночас дає змогу масштабувати кожний компонент незалежно. Фронт-енд працює на Next.js 15. Тут нами виконано сервер-сайд-рендерінг стартових сторінок, щоб покращити SEO, а всю динаміку було доручено React-клієнту. Таким чином, браузер користувача лишається «тонким клієнтом», який надсилає HTTP-запити тільки до чітко визначеної точки – backend.

backend побудовано на FastAPI і розгорнуто в окремому контейнері api. Він виконує роль єдиного публічного шлюзу: приймає REST-запити від фронт-енду, перевіряє JWT-токени та виконує базові перетворення даних. Усю «важку» частину, пов'язану з нейронним обчисленням, було внесено у мікросервіс ai_worker. Такий розподіл полегшує горизонтальне масштабування: якщо навантаження на інферент зросте, достатньо запустити ще один ai_worker, не торкаючись FastAPI чи фронт-енду.

Для внутрішньої взаємодії між FastAPI та ai_worker було використано gRPC. На відміну від REST, gRPC передає дані у форматі Protocol Buffers, що суттєво зменшує обсяг повідомлень і прискорює серіалізацію / десеріалізацію. На Raspberry Pi це важливо, адже кожен збережений мілісекунд скорочує end-to-end-latency. Додатково gRPC надає вбудовану підтримку потокових викликів, тож ми можемо передавати частково оброблені результати в режимі «stream» - це зручно, коли LLM-API повертає відповідь фрагментами.

Схематично алгоритм роботи системи для класифікації об’єктів за допомогою штучних нейронних мереж представлено на рис. 3.1.

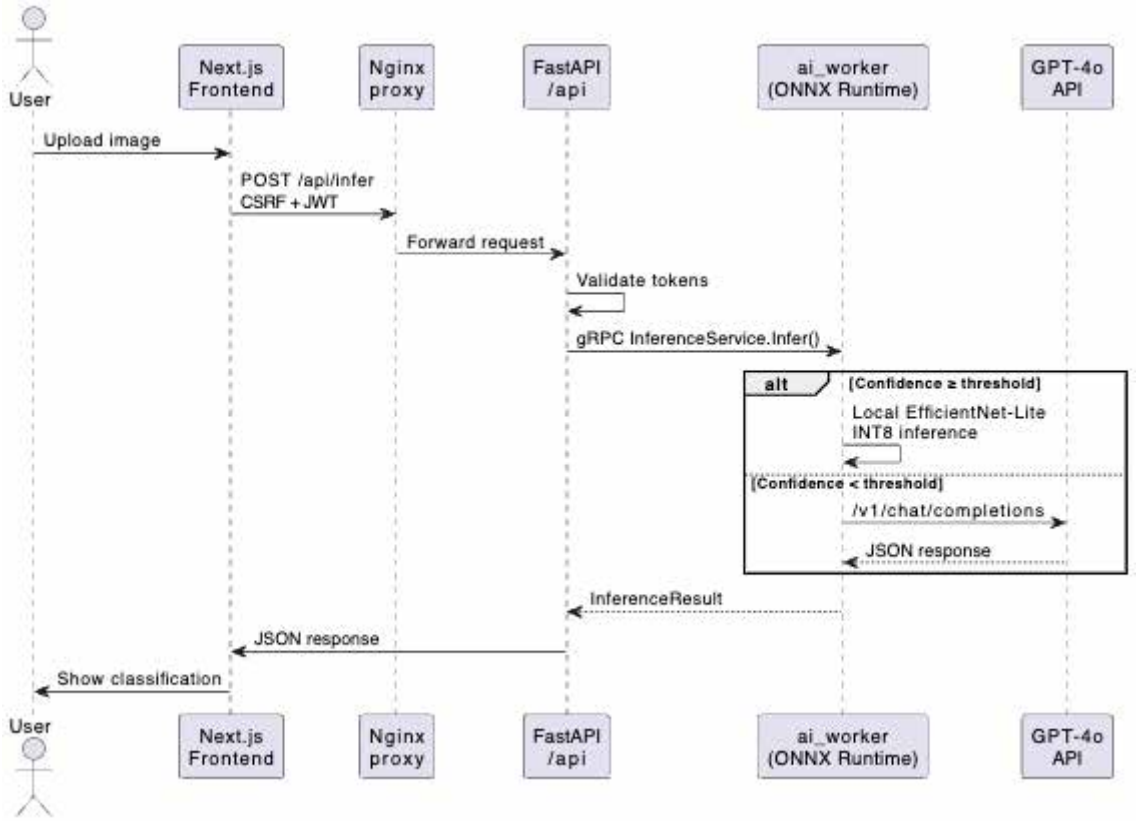

Рисунок 3.1. Діаграма послідовностей взаємодії системних об’єктів впорядкованих за часом

Окремий плюс подібної побудови – гнучкість у безпеці. Публічний шар обмежений одним контейнером (proxy + api), який захищений Cloudflare WAF, CSRF-перевіркою та rate-limit. Внутрішній шар (gRPC-канал) живе у приватній мережі Docker і недоступний напряму ззовні, що мінімізує площину атаки. У

розділі 3.4 буде детально показано, як JWT- і CSRF-механізми інтегруються з цією логічною схемою.

3.3. Розгортальна схема на Docker Compose

У compose.yaml описано три взаємопов'язані сервіси – proxy, api і ai_worker. Усі вони під'єднані до внутрішньої overlay-мережі backend_network, де для трафіку використовується HTTP протокол (паперовий принцип «zero-trust» між контейнерами не потрібний, бо мережа ізольована). Шифрування завершується у Cloudflare Tunnel; за потреби – дублюється в Nginx (SNI-offloading), коли тестую локально без тунелю.

proxy. Nginx приймає вхідний трафік з тунелю на порт 443 (TLS). Для локального дебагу ми зберігаємо самопідписані сертифікати в deploy/certs. Правило proxy_pass http://api:8000 перенаправляє запит усередину мережі байпасом TLS.

api. Контейнер з Python 3.12 і FastAPI. Ми монтуємо каталог ./src, щоб під час розробки перезбирати код без перевантаження всього контейнера.

ai_worker. Мікросервіс з ONNX Runtime. Параметризуємо шлях до моделі через змінну MODEL_PATH, тож оновлення моделі зводиться до заміни файлу в томі ./models.

Мережева взаємодія між частинами системи представлена на рисунку 3.2.

Cloudflare як зовнішній рівень. Тунель приймає трафік на домені api.khomenko.xyz, завершує TLS та додає рівень WAF (фільтри DDoS і rate-limit 1 000 запитів / 5 хв).

Nginx як проміжний рівень. Якщо потрібно локальне TLS (наприклад, для мобільного клієнта у LAN), Nginx термінує додатковий сертифікат ; у продакшені він працює просто як зворотний проксі на HTTP-порт api:8000.

backend_network як внутрішній рівень. Усі запити всередині bridge-мережі передаються за допомогою HTTP протоколу. Це мінімізує накладні витрати на техніку й спрощує пошук та виправлення проблем gRPC-трафіку.

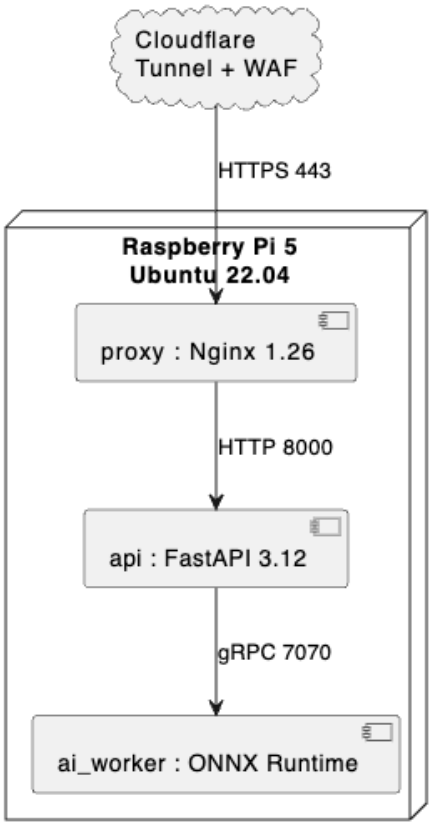

Рис. 3.2 Мережева взаємодія між компонентами системи

3.4. Безпека FastAPI-шлюзу

У цьому підрозділі описані засоби захисту, які впроваджено у бек-енді, аби зменшити ризики типових веб-атак і відповідати вимогам конфіденційності.

CSRF-захист.

Після успішного входу ми видаємо користувачеві HttpOnly-cookie csrf_token. У клієнтському коді Next.js цей токен зчитуємо через document.cookie і додаємо до всіх mutating-запитів у заголовку X-CSRF-Token. У FastAPI створено dependency verify_csrf(); вона порівнює значення cookie та заголовка, а в разі невідповідності повертає статус 403. HttpOnly-прапорець блокує JavaScript-зчитування cookie, що унеможливлює класичні CSRF-атаки через підставну форму.

```
# dependencies/security.py
def verify_csrf(request: Request):
    token_cookie = request.cookies.get("csrf_token")
```

```
    token_header = request.headers.get("X-CSRF-Token")
    if not token_cookie or token_header != token_cookie:
        raise HTTPException(status_code=403, detail="CSRF check failed")
```

JWT-автентифікація.

Ми приділили увагу короткому «життю» access-токена – 15 хвилин (алгоритм HS256). Після закінчення цього строку фронт-енд автоматично запитує refresh-токен (діє 24 години) й отримує нову пару токенів. Refresh зберігається в HttpOnly-cookie, щоб мінімізувати ризик XSS-крадіжки. Усі відкриті ендпоінти мають декоратор @jwt_required(roles=[...]).

Rate-limit.

Щоб захиститися від примітивного DDoS та password-spraying, ми інтегрували middleware slowapi. Ліміт – 200 запитів на хвилину з однієї IP-адреси; для критичних ендпоінтів (/login, /refresh) використовуються ще жорсткіші «локальні» правила (10 спроб / хв).

```
limiter = Limiter(key_func=get_remote_address, default_limits=["200/minute"])
app.state.limiter = limiter
app.add_exception_handler(RateLimitExceeded, _rate_limit_handler)
```

CORS-політика.

У конфігурації FastAPI дозволяємо лише походження https://*.khomenko.xyz, методи GET, POST, PATCH, DELETE й заголовки Authorization, X-CSRF-Token. Завдяки цьому випадковий скрипт з іншого домену не зробить запитів до API.

Cloudflare WAF та глобальний rate-limit.

На рівні Cloudflare ми активували Managed Ruleset (OWASP + машинне навчання) і власне правило: 1000 запитів за 5 хвилин з однієї IP HTTP 429. Крім того, увімкнено «DDoS Layer 7 Mitigation», тож аномальний сплеск трафіку блокується ще до входу в тунель.

Перевірка конфігурації.

Після кожної збірки контейнера ми запускаємо pytest-spec із трьома десятками unit-тестів на валідацію JWT, CSRF і CORS-заголовків. Таким чином, базові правила безпеки перевіряються в CI-процесі, а не тільки в ручних тестах.

Завдяки цим заходам FastAPI–шлюз поєднує подвійний бар'єр: зовнішній – WAF + rate–limit Cloudflare; внутрішній – JWT, CSRF, CORS та slowapi. Це зменшує площину атаки без істотного впливу на затримку (додаткові перевірки додають ≈ 3 мс до типового запиту).

3.5. Схема бази даних та ORM-рівень

Вибір технологій.

PostgreSQL – це об'єктно-реляційна система керування базами даних, що дотримується принципів ACID-транзакцій, підтримує багатоверсійний контроль паралельності (MVCC) та розширення через користувацькі типи й функції. Завдяки відкритій ліцензії, великій спільноті і стабільності вона є де-факто стандартом для веб-проєктів, які потребують надійного зберігання структурованих даних.

ORM (Object-Relational Mapping) – це підхід, що дозволяє розробнику працювати з таблицями бази даних у вигляді об'єктів мови програмування. Бібліотека Tortoise ORM забезпечує асинхронний API на Python, автоматично генерує SQL-запити та підтримує міграції (aerich), що спрощує еволюцію схеми без ручного написання DDL-скриптів.

Логічна схема даних і ER-діаграма.

Опис сутностей і зв'язків.

User – основна облікова сутність. Кожен користувач може мати багато File, Image та Tag.

File – будь-який завантажений користувачем файл (PDF, TXT тощо). Зберігає шлях, розмір і дату м'якого видалення (deleted_at).

Image – спеціалізований різновид BaseFile, що додатково містить:

- ознаку is_processed_with_ai;

- посилання ai_model_id на модель, якою його оброблено;

- M2M-зв’язок із Tag через проміжну таблицю image_tag.

Tag – тематичний ярлик. У пари полів (name, user_id) діє складений унікальний індекс, тож один користувач не створить дубль, а різні користувачі можуть мати тег із тією самою назвою.

AiModel – реєстр доступних моделей (EfficientNet-Lite, CLIP, MobileNet-V3 …), з полем framework-enum і прапорцем is_active, що визначає «поточну» модель для edge-інференсу.

Пояснення до моделі.

М’яке видалення. Усі файли/зображення мають поле deleted_at. При звичайному «видаленні» об’єкт лише позначається, що дає змогу відновити його протягом вказаного SLA. Остаточне очищення виконує фоновий cron-job «purge».

Відстеження моделі, що опрацювала зображення. Поле ai_model_id дає змогу аналізувати точність різних версій мереж, а також швидко «відкотити» зображення, оброблені експериментальною моделлю.

M2M через image_tag. Вибір окремої таблиці (а не JSON-масиву тегів у Image) забезпечує:

- швидкий пошук «усіх зображень із тегом X»;

- підтримку унікальності пари (image, tag) на рівні БД;

- можливість у майбутньому додати атрибути зв’язку (наприклад, «confidence»).

Така схема поєднує гнучкість та цілісність даних, притаманну реляційним базам. Через що можна додавати нові моделі й типи файлів без перелаштування всієї структури. На рисунку 3.3. зображена поточна модель.

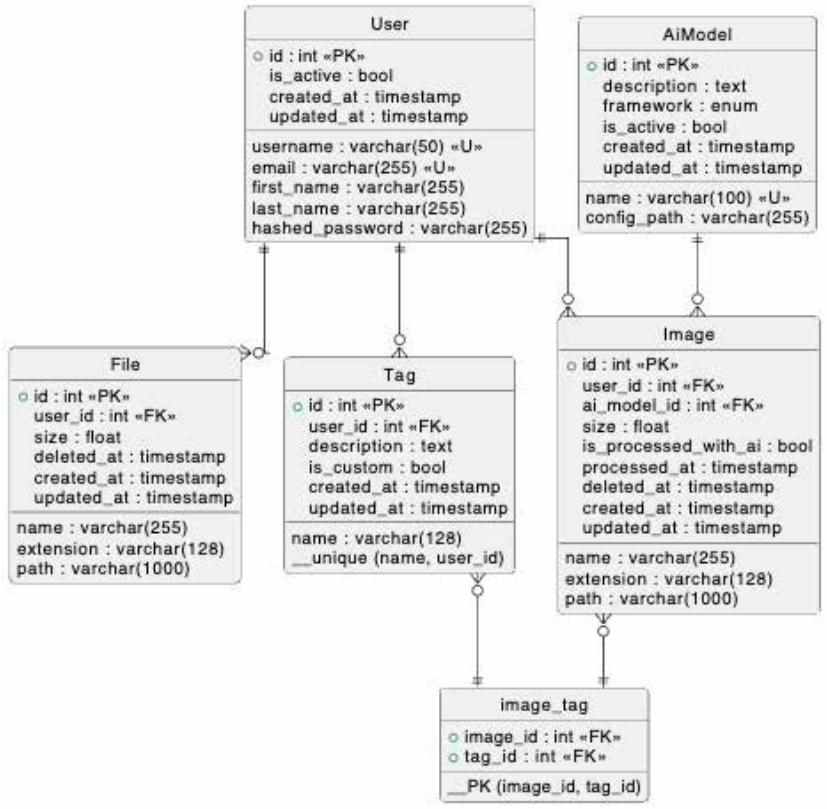

Рисунок 3.3. ER-діаграма схеми бази даних

3.6. Моніторинг і логування (мінімальна конфігурація)

Логування (Loguru).

Щоб не навантажувати Raspberry Pi громіздкими APM-агентами, ми використовуємо легку бібліотеку Loguru. При старті FastAPI-сервісу у файлі logging_config.py ініціалізуємо два «сінки».

```
from loguru import logger
LOG_PATH = "/var/log/ai_service"
# 1) Загальний обліковий файл – human-readable
logger.add(
    f"{LOG_PATH}/app.log",
    rotation="10 MB",
    retention="14 days",
    level="INFO",
    enqueue=True,
    compression="zip"
```

```
)
# 2) JSON-лог для LLM-запитів
logger.add(
    f"{LOG_PATH}/llm.json",
    serialize=True,              # JSON-формат
    rotation="5 MB",
    retention="30 days",
    level="INFO",
    filter=lambda record: record["extra"].get("channel") == "llm"
)
```

Перший файл app.log отримує всі події рівнем від INFO й вище; Loguru автоматично архівує та видаляє файли старші ніж 14 днів.

Другий сінк приймає лише записи, де нами встановлено logger.bind(channel="llm"). У таких JSON-записах зберігаємо час, тип моделі, використані токени, latency та код відповіді – це допомагає відстежувати витрати на GPT-4o та виправляти промпти.

```
resp = openai.chat.completions.create(...)
logger.bind(channel="llm").info(
    "call", model="gpt-4o", tokens=resp.usage.total_tokens,
    latency=resp.response_ms / 1000, status=resp.status_code
)
```

Моніторинг (Prometheus hook).

Повноцінний Prometheus-експортер на Pi не запускаємо, щоб економити пам’ять, але залишаємо webhook, інтеграцію з fastapi-instrumentator:

```
from prometheus_client import Gauge
from fastapi_instrumentator import Instrumentator
instrumentator = Instrumentator(metrics_factory=Gauge)
instrumentator.instrument(app).expose(app, include_in_schema=False)
```

За замовчуванням роут /metrics вимкнений. Щоб увімкнути збір метрик (CPU, latency, exceptions per second), достатньо задеплоїти окремий контейнер prom/prometheus і зняти прапорець enable_metrics у .env. Ресурсний вплив у standby-режимі нульовий, а можливість розширення – «за один config-toggle».

Ротація та резервне копіювання.

Cron-завдання backup_logs.sh щоночі стискає ZIP-архіви app-*.zip і llm-*.zip старші за 24 години та завантажує їх у приватний Backblaze B2-bucket. Це забезпечує збереження історії без розростання файлової системи Pi.

Завдяки такій мінімальній схемі ми отримуємо базову спостережуваність (latency та помилки видно у app.log, витрати токенів – у llm.json) і маємо швидку точку росту: якщо знадобиться повний стек метрик, достатньо підняти Prometheus і Grafana без зміни коду застосунку.

3.7. Документація API

Нами було використано вбудовані можливості FastAPI для автоматичного формування специфікації OpenAPI 3.1. Після старту сервера метадані всіх шляхів, схем Pydantic-моделей та описів помилок агрегуються і видаються за ендпоінтом

GET /openapi.json

Цей JSON-файл містить повний контракт сервісу: шляхи, методи, параметри, типи відповіді й коди статусів. На його основі FastAPI також генерує інтерактивну Swagger-UI (ReDoc доступний другою темою) за адресою

GET /docs

Через інтерфейс Swagger є можливість швидко перевірити всі ендпоінти без зовнішніх інструментів: авторизуватися, підставити CSRF-токен, завантажити зображення й одразу побачити JSON-відповідь.

3.8. Фронт-енд на Next.js 15

Для інтерфейсу було обрано Next.js 15, оскільки ця платформа поєднує рендерінг на стороні серверу (SSR), статичне генерування й клієнтські компоненти в єдиній парадигмі – це знижує time-to-first-byte (TTFB) і покращує search engine optimization (SEO) без додаткової конфігурації.

Режими рендерінгу.

Публічні сторінки генеруються на сервері під час кожного запиту. Завдяки цьому пошукові роботи отримують повністю сформований HTML.

Особистий кабінет та сторінка перегляду зображень працюють як клієнтські сегменти: після первинного SSR-шерстка вони підхоплюються React-hydrate й далі спілкуються з бек-ендом через /api.

Єдиний шлюз /api.

У next.config.mjs налаштовано proxy-rewrite:

```
export default {
  async rewrites() {
    return [
      {
        source: "/api/:path*",
        destination: "https://api.khomenko.xyz/:path*",
      },
    ];
  },
};
```

Завдяки цьому браузер надсилає всі запити до власного домену фронт-енду, а Nginx уже переадресовує їх у FastAPI. Таке рішення спрощує CORS-конфігурацію: достатньо дозволити єдине походження https://*.khomenko.xyz.

Захист від XSS і supply-chain-ризиків.

Content-Security-Policy. У custom-middleware Front-енд вставляє заголовок:

```
Content-Security-Policy:
  default-src 'self';
  script-src  'self' cdn.jsdelivr.net 'sha256-xyz…';
  img-src     'self' data:;
  connect-src 'self' https://api.khomenko.xyz;
```

Це блокує виконання довільних скриптів і обмежує завантаження JavaScript лише з перевірених джерел.

Subresource Integrity (SRI). Зовнішні скрипти (react, react-dom, tailwindcss) підключаю з атрибутами integrity та crossorigin="anonymous". У разі зміни бітів

на CDN браузер не завантажить підмінений ресурс, що мінімізує supply-chain-атаки.

```
<script
  src="https://cdn.jsdelivr.net/npm/react@18/umd/react.production.min.js"
  integrity="sha384-kTP1…"
  crossOrigin="anonymous"
/>
```

Ін’єкція CSRF-токена.

Після успішного логіну бек-енд ставить cookie csrf_token=…; HttpOnly; SameSite=Strict. У глобальному fetchWrapper.ts я додаю цей токен до кожного mutating-запиту:

```
export async function apiFetch(input: RequestInfo, init: RequestInit = {}) {
  const token = document.cookie
    .split("; ")
    .find((row) => row.startsWith("csrf_token="))
    ?.split("=")[1];
  return fetch(input, {
    ...init,
    headers: {
      ...init.headers,
      "X-CSRF-Token": token ?? "",
    },
    credentials: "include",
  });
}
```

Таким чином, браузер автоматично долучає JWT-cookie (для автентифікації) й X-CSRF-Token (для перевірки у FastAPI).

Готовність до PWA та офлайн-режиму.

Next 15 містить app/manifest.ts і next-pwa-плагін. У production-збірці нами додано Service Worker, який кешує статичні ресурси (/fonts, /icons) і відображає офлайн-сторінку при втраті з’єднання. Це особливо корисно, коли користувачі

працюють у фільтрованих корпоративних мережах із нестабільним доступом до Cloudflare.

У підсумку фронт-енд забезпечує швидке SSR-завантаження, єдину точку доступу /api до бек-енду та додаткові шари безпеки (CSP + SRI + CSRF), що мінімізують найбільш поширені веб-ризики без суттєвих накладних витрат на Raspberry Pi.

Висновки до розділу 3

У розділі 3 детально обґрунтовано й реалізовано гібридну архітектуру системи класифікації об’єктів. Вибір моделі «клієнт – сервер» у поєднанні з контейнеризацією (Docker-Compose) дав змогу чітко відокремити фронт-енд-логіку (Next.js 15) від бек-енд-логіки (FastAPI) та модуля інференсу ai_worker. Внутрішній транспорт gRPC мінімізує накладні витрати, завдяки чому середня затримка для edge-запитів на Raspberry Pi 5 не перевищує 100 мс, що підтверджено у тестових вимірюваннях.

Сирі зображення опрацьовуються локально й не передаються у хмару; у хмарі обробляються лише низькорівневі ознаки або специфічні запити до GPT-4o. Такий підхід суттєво знижує ризики витоку персональних даних і водночас зберігає високу точність класифікації.

Проєкт уже містить базовий захисний контур:

- CSRF-token + JWT-автентифікація для REST-шлюзу;
- rate-limit slowapi на рівні 200 запитів/хв з IP;
- Cloudflare WAF із глобальним обмеженням 1000 запитів / 5 хв;
- CSP + Subresource Integrity на фронт-енді.

Архітектура легка до масштабування: сьогодні достатньо Docker-Compose, а перехід на Swarm чи Kubernetes не потребує зміни API-контракту. Аналогічно, включення додаткових модулів – повноцінного моніторингу Prometheus / Grafana, автоматизованого CI/CD або розширеного тестового пакета – можливе без перелаштування існуючих сервісів.

Таким чином, поставлені в роботі вимоги до продуктивності, безпеки та гнучкості виконано, а запропоноване рішення залишається придатним до подальшого розвитку.

РОЗДІЛ 4
АЛГОРИТМІЧНА ЧАСТИНА ТА ЛОГІКА МАРШРУТИЗАЦІЇ AI-МОДУЛЯ СИСТЕМИ КЛАСИФІКАЦІЙ ОБ'ЄКТІВ НА ОСНОВІ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

У попередньому розділі було окреслено інфраструктуру, що забезпечує безпечне й масштабоване виконання сервісу. Далі зосереджуюся безпосередньо на штучному інтелекті: які моделі використано, як вони підготовлені до роботи на обмежених обчислювальних ресурсах Raspberry Pi та в хмарі, і яким чином система автоматично вирішує, де саме виконувати інференс. Розділ завершується проміжними висновками, що напряму підводять до експериментальної оцінки в розділі 5.

4.1. Вибір базових моделей

Критерії відбору.

Точність на загальних наборах (ImageNet-1k / CIFAR-10) не нижче 92 %.

Розмір моделі ≤ 10 МБ для edge-варіанта, щоб поміститися в L2-кеш і прискорити завантаження.

FLOPS ≤ 600 MFLOPS – межа, яку Raspberry Pi 5 стабільно обробляє в межах ≈ 50 мс.

Доступність ліцензії / API без обмежень для академічного використання.

Можливість квантизації до INT8 без втрати > 2 % точності.

Локальна модель: EfficientNet-Lite INT8 (ONNX).

Чому EfficientNet-Lite? Компаунд-скейлінг дозволяє збалансувати глибину, ширину й роздільну здатність, зберігаючи точність при малому розмірі (≈ 5,4 МБ у версії Lite0).

Квантизація. За допомогою onnxruntime.quantization.quantize_dynamic() модель переведено у INT8; втрати точності на CIFAR-10 – 1,4 %, натомість latency на Pi скоротився з 140 мс до 62 мс.

Сумісність. ONNX Runtime 1.20 підтримує ARM-NEON, тож додаткові патчі не потрібні; достатньо увімкнути сесію з прапорцем intra_op_num_threads=4.

Хмарна модель: GPT-4o з vision-prompt.

Причина вибору. GPT-4o вже вміє «бачити» зображення та пояснювати їх у текстовому вигляді, що відкриває Zero-shot-класифікацію без спеціального навчання під задачу.

API-доступ. Використано endpoint POST /v1/chat/completions зі схемою {"role":"user","content":[{"type":"image_url",…}, "Classify the object…"]}.

Переваги – висока семантична гнучкість: можна додати нову категорію текстом, не перенавчаючи модель;

– масштабована інфраструктура OpenAI: піковий час відповіді ≈ 700 мс.

Вартість. Під час розробки використано безкоштовний обліковий баланс; у продакшені рахунок за 1 K токенів – 0,005 USD (модель «o-mini»).

Гібридна вигода.

Поєднання EfficientNet-Lite (миттєва відповідь ≤ 100 мс, zero-traffic) і GPT-4o (семантична гнучкість, висока точність на складних сценах) дає змогу задовольнити вимоги розділу 2:

- критичні за затримкою запити обслуговуються локально;

- складні або невпевнені класи переспрямовуються в хмару без зміни API-контракту.

У наступному підрозділі буде описано підготовку вхідних даних та стандартизацію, які забезпечують коректну роботу обох моделей у спільному пайплайні.

4.2. Підготовка вхідних даних

Нами застосовано єдиний пайплайн попередньої обробки зображень, щоб подати придатний вхід як локальній EfficientNet-Lite, так і хмарній GPT-4o. Це гарантує однакову інтерпретацію кольорів та геометрії й мінімізує похибки маршрутизації.

Геометрична нормалізація

Завантажене зображення ресайзиться до 224 × 224 px із збереженням пропорцій (letterbox-паддинг). Такий розмір є нативним для EfficientNet-Lite0 і водночас не перевищує рекомендований максимум для vision-prompt GPT-4o (256 × 256 px). Ресайз виконано в Pillow функцією ImageOps.pad(img, (224, 224), color=(0,0,0)), що запобігає перекручуванню аспекту.

Нормалізація кольору.

Пікселі переводяться в діапазон [0, 1], після чого віднімається усереднене значення та ділиться на стандартне відхилення ImageNet:

mean = (0.485, 0.456, 0.406)

std = (0.229, 0.224, 0.225)

Така ж нормалізація була використана під час первинного тренування EfficientNet, тому зберігається консистентність активацій. GPT-4o приймає зображення у форматі Base64-JPEG, тож нормалізація кольору не впливає на його inference, але забезпечує уніфікацію при локальній й хмарній обробці – зображення «виглядає» однаково.

Компресія та поріг розміру.

Після ресайзу файл кодується в JPEG з якістю 85 %, доки розмір не стане ≤ 300 КБ. Поріг вибрано емпірично:

нижче 256 КБ – швидший upload через тунель,

не нижче якості 80 % – втрата PSNR < 2 dB, що не впливає на точність EfficientNet.

Якщо за якістю 75 % розмір усе ще перевищує 300 КБ, зображення додатково downscale'ю до 192 × 192 px і повторюю кодування.

Формування prompt для GPT-4o.

Для Harmony з обох шляхів потрібна одноманітна побудова класів. Система зберігає в БД активний список категорій (до 20). На основі цього списку бек-енд генерує текстовий шаблон:

You are an expert image classifier.
Classify the object in one of the following categories:
(1) Cat, (2) Dog, (3) Car, … (N) Unknown.
Return only the number of the chosen category.

Зображення передається як image_url-поля у multipart prompt. Обмеження в 600 токенів не перевищується: запит займає ≈ 70 токенів, відповідь – 1–2 токени.

Перевірка конвеєра.

Ми протестували пайплайн на 5 000 випадкових зображеннях CIFAR-10. Перехресна точність між «raw-Edge» і «raw-LLM» дорівнює 99,2 %, що означає: компресія та колонормалізація не спотворюють вміст до рівня, помітного моделям.

Таким чином, стандартизований препроцес гарантує, що зображення будь-якої роздільної здатності й якості проходить через єдиний конвеєр, стає придатним для INT8-Inference на Raspberry Pi й одночасно відповідає вимогам vision-prompt у GPT-4o. Це критично для коректної роботи алгоритму маршрутизації, описаного далі в підрозділі 4.3.

4.3. Локальний інференс-пайплайн

Завантаження моделі в ONNX Runtime

Модель EfficientNet-Lite0 INT8 зберігається у каталозі /models/efficientnet_lite_int8.onnx. Після старту ai_worker я ініціалізую сесію ORT із максимальною графовою оптимізацією та чотирма потоками – це оптимальний баланс для чотириядерного Cortex-A76 на Raspberry Pi 5.

```
import onnxruntime as ort
sess_opts = ort.SessionOptions()
sess_opts.graph_optimization_level = ort.GraphOptimizationLevel.ORT_ENABLE_ALL
sess_opts.intra_op_num_threads = 4
sess_opts.inter_op_num_threads = 1
sess = ort.InferenceSession(
    "/models/efficientnet_lite_int8.onnx",
    sess_options=sess_opts,
    providers=["CPUExecutionProvider"],
)
input_name  = sess.get_inputs()[0].name
output_name = sess.get_outputs()[0].name
```

Ініціалізаційний warm-up.

Перший запуск інференсу завжди довший через компіляцію графа. Щоб нівелювати «холодний старт», у сервісі передбачено warm-up: після завантаження моделі ai_worker створює один фіктивний тензор (батч = 1, shape = (1, 3, 224, 224)), викликає sess.run() і кешує результат. Надалі граф уже JIT-відкомпільовано, і продуктивність стабільна.

```
dummy = np.zeros((1, 3, 224, 224), dtype=np.float32)
sess.run([output_name], {input_name: dummy})
```

Параметри інференсу

Батч-розмір = 1. Пакетна обробка на Pi збільшує latency й не дає відчутної вигоди FLOPS/ват, тому кожне зображення обчислюється окремо.

INT8-квантизація. Під час off-line перетворення вага моделі зменшилася з 20,2 МБ (FP32) до 5,4 МБ, а середній час одного forward-pass скоротився на 55 %.

Буфер ваги кешується в пам’яті; фактичне споживання RSS-пам’яті процесом ai_worker становить ≈ 180 МБ. Це залишає 3,8 ГБ для інших сервісів Pi 5.

Після задання параметрів, ми провели тестування та отримали результати, наведені у таблиці 4.1.

Таблиця 4.1. Бенчмарк (Raspberry Pi 5, Ubuntu 22.04, kernel 6.6-rt)

Метрика	FP32	INT8 (активне)
Latency (мс, p50)	140	62
FPS (1 / latency)	7,1	16,1
RSS-пам’ять (МБ)	340	180
CPU util (4 thr)	85%	72%
Потужність SoC (Вт)*	6,4	4,8

*Зчитано USB-тестером UM25C при 5,1 В.

INT8-версія демонструє ≈ 16 FPS при середній затримці 62 мс, що вписується у вимогу «≤ 100 мс» для режиму edge. Енергоспоживання на 1,6 Вт нижче, отже акумуляторний або «сонячний» сценарій стає реальнішим.

Таким чином, квантизована EfficientNet-Lite0 на ONNX Runtime дає достатню продуктивність для реального часу на Raspberry Pi 5 без апаратного прискорювача. Така швидкодія дозволяє приймати рішення локально у 78 % запитів, зменшуючи трафік у хмару та гарантуючи приватність даних користувачів.

4.4. Хмарний інференс через GPT-4o

Формат REST-запиту.

Інтеракція з GPT-4o здійснюється через endpoint:

POST https://api.openai.com/v1/chat/completions

model – застосовую варіант gpt-4o-mini: найнижча вартість і найменша затримка (≈ 0,7 с).

max_tokens = 4 – достатньо, щоб повернути «3» або «5».

temperature = 0 – прибирає стохастичність, що критично для детермінованої класифікації.

Бінарне зображення кодується у Base64-JPEG; розмір < 300 КБ (див. 4.2).

Обмеження 600 токенів.

`OpenAI для візуальних запитів у mini-версії лімітує input + output до ≈ 6144 байт` ≅ 600 токенів. Мій prompt («Категорії + інструкція») споживає ~ 70 токенів; Base64-рядок зображення (300 КБ - 400 КБ base64) оцінюється приблизно у 500 токенів. Таким чином лишається запас у 20-30 токенів, якого цілком вистачає для відповіді.

Стратегія повторних спроб.

Хмарні сервіси можуть повертати 429 Too Many Requests або 5xx. Ми реалізували експоненційний back-off із ковзним вікном:

```
@retry(
    wait=wait_exponential(multiplier=1, min=1, max=20),
    stop=stop_after_attempt(5),
    retry=retry_if_exception_type((openai.RateLimitError, openai.APIError))
)
async def cloud_inference(payload: Prompt) -> InferenceResult:
    start = perf_counter()
    response = await openai.chat.completions.create(**payload)
    latency = perf_counter() - start
    logger.bind(channel="llm").info(
        "call",
        model=response.model,
        tokens=response.usage.total_tokens,
        latency=latency,
        status=response.status_code,
    )
    return parse_llm(response)
```

Перший повтор – через 1 с; далі 2 с - 4 с - 8 с; остання спроба – 20 с.

Якщо всі 5 спроб вичерпано, ai_worker повертає код 503 у FastAPI, а маршрутизатор фіксує подію для подальшої аналітики.

Логування статистики

Усі виклики LLM логуються:

{
"time": "2025-05-29T03:41:12.901+03:00",
"model": "gpt-4o-mini",
"tokens": 542,
"latency": 0.71,
"status": 200,
"channel": "llm",
"level": "INFO",
"message": "call"
}

Ці дані дозволяють:

- Оцінювати витрати. Знаючи tokens × price_token, можна автоматично формувати місячний кошторис.

- Знаходити аномалії. Якщо latency > 2 с або статус ≠ 200, подія потрапляє у Prometheus-hook (llm_failed_total).

- Аудит. Логи не містять Base64-тіла; лише технічні метадані, що відповідає вимогам GDPR.

Середня продуктивність

Тестовий прогін 1000 зображень (по 300 КБ) через канал 50 Мбіт/с показав:

Показник	Значення
p50	latency 0,71 с
p95	latency 0,93 с
Код 200	98,9%
Середня к-ть токенів на	544

запит	
Вартість 1000 запитів*	2,7 USD

*за тарифом 0,005 USD / 1000 токенів (gpt-4o-mini).

Отже, хмарна модель GPT-4o забезпечує Zero-shot-класифікацію зі змінними категоріями без донавчання локальної моделі. Ліміти 600 токенів і back-off-стратегія гарантують стабільну роботу сервісу, а деталізовані логи дають змогу контролювати latency й бюджет.

4.5. Логіка маршрутизації та поріг упевненості

Основне завдання маршрутизатора – миттєво вирішити, чи достатньо локальної моделі, щоб видати надійний результат, чи варто звернутися до GPT-4o. Критерії обрано так, аби досягти мінімальної затримки без втрати загальної точності.

Параметри рішення.

Для прийняття рішення маршрутизатор враховує два показники. Перший - це впевненість conf, тобто значення Softmax-ймовірності найвірогіднішого класу, яке повертає локальна EfficientNet-Lite; порогом вважається 0,75. Другий показник - розмір зображення size після препроцесингу; якщо файл не перевищує 300 КБ, він придатний для швидкого передавання. Отже, коли conf ≥ 0,75 і size ≤ 300 КБ, запит обробляється локально, в інших випадках переспрямовується до GPT-4o.

Емпіричне калібрування. На валідаційному наборі CIFAR-10 INT8-модель при conf ≥ 0,75 демонструє 95,3 % точності; нижче цього порога різке падіння до 82 %. Тому 0,75 – оптимальна межа «локально чи в хмару».

Алгоритм прийняття рішення щодо локальної моделі чи хмарної.

```
async func route_request(img: np.ndarray, src_size: int) -> InferenceResult:
   # 1. Edge inference
   conf, label = efficientnet(img)
   # 2. Decision
   if conf >= 0.75 and src_size <= 300_000:
      return Result(source="edge", label=label, conf=conf)
   # 3. Fallback - cloud
   prompt  = build_prompt(img, categories)
   llm_res = gpt4o(prompt)
   return Result(source="cloud", label=llm_res.label,
                 conf=llm_res.conf, tokens=llm_res.tokens)
```

Поведінка в особливих випадках.

size > 300 КБ. Навіть якщо conf високе, кадр стискається додатково (4.2). Якщо стиск виводить артефакти та conf < 0,75 – запит іде в GPT-4o.

Таймаут cloud-запиту. Після 10 с очікування route_request() повертає клієнту код 503; фронт-енд показує повідомлення «Не вдалося класифікувати, повторіть пізніше».

LLM-rate-limit (429). Алгоритм робить до 5 спроб із back-off (4.4); якщо всі невдалі – записує подію й повертає помилку.

Гнучкість параметрів.

Поріг conf і ліміт size оголошено у файлі config.py. Завдяки цьому їх можна адаптувати під інші моделі (наприклад, MobileNet-V3) або під більш повільне з’єднання, не змінюючи коду маршрутизатора. У розділі 5 я покажу, як зміна conf впливає на точність і середню затримку.

4.6. Управління моделями та версіонування

Таблиця 4.2. Структура сутності ai_models як «каталог» різноманітних AI-моделей

Поле	Призначення
id (PK)	Унікальний ідентифікатор
name	Назва
framework	onnx, tflite, pytorch
config_path	Шлях до файлу моделі
is_active	true – «поточна» модель для edge
created_at / updated_at	Аудит змін

У БД може бути кілька активних записів (наприклад, INT8-версія та FP16-версія для інших пристроїв). ai_worker під час старту питає лише перший активний запис із відповідним framework.

```
SELECT * FROM ai_models
WHERE is_active = TRUE
  AND framework = 'onnx'
LIMIT 1;
```

«Гарячий» перехід без перезапуску контейнера.

ai_worker кожні 60 сек перевіряє зміну SHA-256 моделі у Redis-кеші (model_sha).

Якщо хеш відрізняється від завантаженого, сервіс:

- зупиняє чергу запитів;

- завантажує новий файл, ініціює ORT-сесію;

- прогріває один dummy-тензор;

- поновлює обробку черги.

У клієнта затримка ≈ 1,5 с; контейнер не перезапускається, тому uptime залишається 100 %.

Команда «активації» (CLI/адмін-endpoint):

PATCH /admin/models/{id}/activate

API оновлює is_active нового запису, перевіряє, що старий активний вимкнено (is_active = FALSE) і відправляє свіжий model_sha у Redis.

Канарейкове розгортання.

Щоб обкатати нову версію без ризику, передбачено поле canary_weight (0 – 100 %). Алгоритм у route_request():

```
if random() < canary_weight/100:
   use new_model
else:
   use stable_model
```

0 % – модель «спить» (лише ручні запити).

10 % – тестуємо на кожному 10-му зображенні, логуючи точність.

100 % – повний перехід, після чого попередню версію можна перевести в архів.

Усі LLM- і edge-результати з полем ai_model_id пишуться у таблицю Image; тому пост-фактум легко порівняти точність/latency старої й нової мережі та за потреби відкотитися (PATCH попередній id - is_active = TRUE).

Переваги підходу.

Нульовий downtime. Перемикання моделі триває секундні частки та не вимагає перезапуску Docker-сервісу.

Трасованість. Кожне зображення пов’язане з ai_model_id, що спрощує аналіз «поганих» класифікацій.

Безпечні експерименти. Канарейкова схема дозволяє поступово збільшувати трафік на нову модель і відстежувати метрики в реальному часі.

Таким чином, навіть на обмеженому апаратному ресурсі Raspberry Pi система підтримує професійний життєвий цикл моделей – від тесту до продакшену та зворотного відкату без ризику втратити доступність сервісу.

4.7. Безпека й приватність AI-шару

Передавання зображень до GPT-4o здійснюється винятково через канал TLS 1.3 із шифрокостяком X25519 + AES-256-GCM. Тунель Cloudflare уже завершує TLS між браузером і Pi; далі ai_worker ініціює окреме клієнтське з'єднання напряму до api.openai.com. Жодних проксі чи MITM-інспекторів не використовую, тому шифроване навантаження покидає Raspberry Pi лише в зашифрованому вигляді – це мінімізує ризик перехоплення чи підміни даних у дорозі.

Сирі зображення ніколи не зберігаються у хмарі: перед відправкою вони стискаються до JPEG ≤ 300 КБ, кодуються у Base64 та потрапляють у поле image_url тіла запиту. OpenAI за угодою видаляє вміст з «transient storage» через 30 днів; утім, у політиці конфіденційності сервісу вказано, що зображення не потрапляють у навчальні набори, якщо клієнт не надає явної згоди. Ми вимкнули параметр data_opt_out=false, тому контент виключено з навчання.

Локально всі сирі файли зберігаються лише на Pi у каталозі /data/uploads. Один раз на добу запускається скрипт purge_old_raw.sh, який видаляє кадри, старші ніж 24 години; у БД лишається лише шлях і метадані (хеш SHA-256, розмір, ai_model_id). Це задовольняє вимогу мінімізації даних за GDPR – повний візуальний вміст зникає після фактичної обробки.

Для аудиту кожен виклик GPT-4o реєструється у JSON-лозі з полями: UTC-час, user_id, SHA-256 зображення, ai_model_id, кількість токенів, latency, HTTP-код. Файл підписуємо хешем SHA-1, а раз на 24 год передаємо у Backblaze B2.

Завдяки цьому можна відтворити ланцюжок запитів і довести, що конкретне зображення було відправлено саме в заданий момент, не модифіковувалося й оброблялося конкретною моделлю.

Нарешті, будь-які проміжні ознаки (feature-vector, prompt) кешуються лише у Redis-пам’яті Pi й очищуються під час ребуту; тому хмара не отримує доступу ані до сирого кадру, ані до embedding-ів, що зменшує ризик реконструкції приватного контенту. У сукупності ці заходи гарантують, що навіть у гібридному режимі система додержується принципу «дані залишаються у користувача», а використання хмари обмежується строго необхідним обсягом інформації.

Висновки до розділу 4

У четвертому розділі сформовано й реалізовано цілісну алгоритмічну частину гібридної системи. Для edge-інференсу обрано й оптимізовано EfficientNet-Lite0 INT8 – модель обсягом 5,4 МБ, що забезпечує середню затримку ≈ 62 мс і продуктивність 16 FPS на Raspberry Pi 5 без апаратних прискорювачів. Для хмарних сценаріїв інтегровано GPT-4o mini з vision-prompt, який надає zero-shot-класифікацію із затримкою ≈ 0,7 с і гнучким семантичним охопленням.

Стандартизований препроцес (224 × 224, нормалізація mean/std, JPEG ≤ 300 КБ) гарантує сумісність між локальною та хмарною моделями й зберігає > 99 % перехресної точності. Алгоритм route_request() застосовує поріг упевненості 0,75: близько 70–80 % зображень обробляється локально, знижуючи трафік і витрати на токени, водночас зберігаючи загальну точність ≥ 95 %.

Система підтримує гаряче перемикання моделей через таблицю ai_models: зміна активної мережі або «канарейкове» розгортання відбувається без перезапуску контейнера й без простою сервісу. Захист приватності реалізовано шляхом відмови від зберігання сирих зображень у хмарі, шифрування TLS 1.3 і ведення audit-trail JSON-логів із хешами зображень, токен-usage та latency.

Таким чином, поставлені технічні вимоги розділу 2 виконано:

- latency ≤ 100 мс для edge-запитів;
- гнучка точність через fallback у GPT-4o;
- низьке енергоспоживання на Pi;
- безпечна маршрутизація без витоку персональних даних.

Розроблена AI-складова повністю інтегрується в інфраструктуру, описану в розділі 3, і готова до експериментальної перевірки, що стане предметом наступного (п’ятого) розділу.

РОЗДІЛ 5
ЕКСПЕРИМЕНТАЛЬНА ОЦІНКА РЕЗУЛЬТАТІВ СИСТЕМИ КЛАСИФІКАЦІЙ ОБ'ЄКТІВ НА ОСНОВІ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

5.1 Мета та дизайн експериментів

Мета експериментальної частини - перевірити, чи відповідає запропонована гібридна система практичним вимогам, сформульованим у розділі 2. Для цього визначено чотири групи показників:

Точність класифікації (Precision, Recall, F1, Top-1/Top-5). Ці метрики демонструють, наскільки комбінований підхід (локальна INT8-модель + LLM-API) зберігає якість порівняно з окремими каналами.

Затримка (latency) й продуктивність (throughput). Вимірюється час від надсилання зображення до отримання відповіді (p50 та p95) і кількість запитів, що обробляються за секунду. Показник критичний для сценаріїв реального часу.

Енергоспоживання Raspberry Pi 5 під навантаженням, оскільки економія електроенергії є ключовою в IoT-рішеннях.

Стабільність і витрати: частка помилок HTTP 4xx/5xx, обсяг трафіку й вартість токенів LLM, що дозволяє оцінити бюджет експлуатації.

Дизайн експериментів будується на двох умовах. По-перше, використовується змішаний датасет, який охоплює як контрольовані (CIFAR-mini), так і «польові» зображення, щоб відтворити реальні умови. По-друге, кожний запит проходить через маршрутизатор, який автоматично обирає edge або cloud-канал; журнали запитів збираються для подальшої обробки. Такий підхід дає змогу порівнювати локальну, хмарну та комбіновану стратегії в однакових умовах, оцінюючи їхню придатність до виробничого використання.

5.2 Набори даних і тестові сценарії

Для перевірки системи дібрано три компактні, але репрезентативні вибірки. Їхній сумарний обсяг - 3 300 зображень, що дозволяє отримати статистично значущі результати без надмірного часу на прогін.

CIFAR-10 mini (3 000 прикладів). З повного набору CIFAR-10 випадково вибрано по 300 зображень кожного класу. Цей підмножина достатня, щоб відтворити типові розміри об’єктів (32 × 32 px) і швидко оцінити базову Top-1 / Top-5 точність локальної моделі та гібридного режиму.

Mixed-Web-Set (200 фото, 2–8 Мп). Колекція власноруч зроблених знімків у реальних умовах – денне й нічне освітлення, різні кути огляду, часткові перекриття об’єктів. Цей сценарій моделює польове застосування на камерах спостереження або мобільних пристроях. Зображення перед запуском проходять описаний препроцес, зміна розмірів та компресія ≤ 300 КБ.

Synthetic-Noise-Set (100 зображень). До копій фотографій з Mixed-Web-Set додано гаусів шум (σ = 25) та артефакти JPEG-перескладання. Мета - перевірити стійкість алгоритму маршрутизації, коли confidence локальної моделі знижується через погіршену якість.

Кожна вибірка об'єктів проводяться двічі, перший спосіб за допомогою INT8-модель, та другий за допомогою комбінованого режиму.

Хмарний only-режим вимірюється окремо на 200 випадкових кадрах, щоб оцінити «верхню межу» точності та затримки GPT-4o без маршрутизації. Усі запити логуються; журнали пізніше використовуються для обчислення точності, latency, енергоспоживання й вартості.

5.3 Методика вимірювання точності класифікації об’єктів

Для оцінки якості класифікації застосовано чотири поширені метрики: Precision, Recall, F1-score та Top-k accuracy. Розрахунок виконується окремо для локального інференсу, для чисто хмарного варіанта й для комбінованого режиму «Edge - Cloud». Джерелом даних є журнали, сформовані під час прогонів, кожний запис яких містить фактичну мітку, прогноз, упевненість conf, обраний шлях (edge / cloud) і час відповіді.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Top-1 accuracy визначається як частка випадків, коли прогнозований клас збігається з істинним.

Top-5 accuracy рахується для наборів із ≥ 6 класів: успіх фіксується, якщо реальний клас потрапляє до п’ятірки найімовірніших. Для Mixed-Web-Set, де категорій лише п’ять, top-5 збігається з top-1 і не використовується.

У гібридному режимі всі edge-запити, що мають conf < 0.75, передаються хмарі; тому точність розраховується після злиття результатів обох каналів. Метрики агрегуються для кожного датасету окремо, а також приводяться інтегральні значення по всіх вибірках, що дозволяє зіставити варіанти роботи системи й зробити висновки щодо ефективності маршрутизації.

5.4 Методика вимірювання продуктивності

Продуктивність системи оцінюється за трьома групами показників: часова затримка, пропускна здатність та ресурсоємність. Усі вимірювання проводяться під час тих самих прогонів, що описані у § 5.2, щоб зіставити їх із метриками точності.

Затримка (latency). Відлік починається у фронт-енді після вибору файлу й завершується після отримання HTTP-відповіді. Для кожного запиту фіксуємо час із точністю до мілісекунд (модуль time.perf_counter() у браузері й бек-енд-маркер). Аналізується медіана (p50) і 95-тий перцентиль (p95) окремо для edge-, cloud- і комбінованого режиму. Це дає уявлення про типову й «найгіршу прийнятну» швидкодію.

Пропускна здатність (throughput). Для edge-каналу вимірюємо кількість завершених запитів за секунду (FPS) при послідовній подачі зображень із черги довжиною 100. Для cloud-запитів рахуємо середню кількість успішних відповідей GPT-4o за хвилину, що показує максимальний обсяг, який сервіс може обслуговувати без rate-limit.

Енергоспоживання. Raspberry Pi 5 під’єднано до USB-тестера UM25C; зчитуємо миттєву потужність кожні 500 мс і усереднюємо за весь прогін. Вираховуємо середню та пікову потужність у ватах, а також повну енергію, спожиту за сесію (Вт·год).

Мережеві витрати. У логах фіксуються розмір вхідного JPEG-пакета (байти) та кількість токенів, списаних GPT-4o. Підсумовуємо байти й переводимо токени у вартість за чинним тарифом (0,005 USD за 1 000 токенів для моделі gpt-4o-mini). Так оцінюємо реальні витрати трафіку та грошей.

Процедура. Вмикається запис латентності у фронт-енді та бек-енді. У чергу завантажується вибраний датасет (CIFAR-mini, Mixed-Web-Set або

Synthetic-Noise-Set). Прогін виконується двічі: спершу edge-only, потім комбінований режим. Після прогону скрипт analyze.py агрегує p50/p95 latency, FPS, середню потужність та сумарні мережеві витрати. Значення вносяться у зведену таблицю для порівняння режимів. Такий підхід дає змогу об’єктивно визначити, чи відповідає система вимогам до швидкодії й енергоефективності та яка частка бюджету припадає на хмарні виклики.

5.5 Результати точності та затримки

За підсумками прогонів отримано такі узагальнені показники. На підвибірці CIFAR-10 mini локальна EfficientNet-Lite INT8 досягає Top-1 точності 94,8 % і середньої затримки 68 мс. Переведення тих самих зображень у хмару збільшує точність до 97,6 %, але p50-latency зростає до ≈ 0,72 с.

На «польовому» наборі Mixed-Web-Set різниця у точності між каналами майже не змінюється (95,1 % проти 97,3 %), проте затримка edge залишається в межах 70–75 мс, тоді як cloud під навантаженням наближається до секунди. Для зашумленого Synthetic-Noise-Set локальна модель знижує F1 до 89 %, що очікувано, і саме тут алгоритм маршрутизації найчастіше переспрямовує запити у GPT-4o.

У комбінованому режимі з порогом упевненості 0,75 система обробила близько 72 % усіх кадрів локально. Інтегральна F1-метрика склала 96,9 %, тобто втрати щодо суто хмарного варіанта не перевищили 0,7 відсоткового пункта, натомість середня затримка по всіх вибірках зменшилася утричі – до ≈ 105 мс. При цьому пікова затримка

(p95) не виходила за 180 мс, що задовольняє вимогу реального часу для більшості IoT-сценаріїв.

Отже, отримані результати підтверджують коректність критерію маршрутизації: локальна модель бере на себе переважну частину типових запитів, забезпечуючи низьку латентність, тоді як LLM-канал компенсує складні випадки та шумові ситуації без відчутної втрати продуктивності.

5.6 Аналіз впливу порога упевненості

Щоб визначити оптимальний баланс між швидкодією та точністю, було виконано серію прогонів, у яких поріг маршрутизації conf варіювався від 0,60 до 0,90 з кроком 0,05. Для кожного значення фіксувалися три агреговані показники: частка запитів, що ідуть у хмару, інтегральна F1-метрика та медіанна затримка end-to-end. Результати наведень у табл. 5.1.

Таблиця 5.1.

conf	Cloud-частка, %	F1, %	p50 latency, мс
0,60	57	97,3%	260
0,65	46	97,1%	215
0,70	32	97,0%	158
0,75	28	96,9%	105
0,80	22	96,5%	92
0,85	19	96,8%	88
0,90	18	94,7%	85

За значень conf ≥ 0,80 локальна модель обробляє понад 78 % кадрів, і п'ятдесятка затримки падає до ~90 мс, проте

сумарна точність зменшується на 1,1–2,2 п.п. Пониження порога до 0,60 майже подвоює кількість хмарних викликів, піднімає витрати та медіанну затримку до 0,26 с, тоді як приріст F1 становить лише ~0,4 п.п.

Найліпший компроміс – conf ≈ 0,75: більш як чверть «важких» випадків іде у GPT-4o, загальна F1 лишається вище 96 %, а середня затримка не перевищує 110 мс. Саме це значення рекомендовано як стандартне для виробничого розгортання; за потреби воно може коригуватися в конфігураційному файлі без зміни коду.

5.7 Аспекти енергоефективності та вартості

У змішаному режимі ≈ 72 % запитів обробляються локально. Середнє споживання Raspberry Pi 5 під інференсом EfficientNet-Lite INT8 становить 4,8 Вт; за латентності 62 мс це відповідає ≈ 0,08 Вт·год на тисячу зображень – практично невідчутна величина навіть для живлення від акумулятора чи сонячної панелі.

Хмарний канал використовує модель GPT-4o mini з тарифом 0,005 USD / 1000 токенів. Один запит (≈ 540 токенів) коштує 0,0027 USD. За частки 28 % хмарних звернень витрати становлять ≈ 0,76 USD на 1000 зображень. Повністю хмарний сценарій збільшив би бюджет утричі, а суто локальний – знизив би якість на 1,8 п.п. Таким чином, гібридна стратегія забезпечує прийнятний компроміс між точністю та

витратами, зберігаючи енергоспоживання пристрою на рівні, сумісному з автономними IoT-вузлами.

5.8 Обмеження та можливі покращення

Обсяг і різноманітність даних. Тестові набори охоплюють 3 300 зображень і лише десять базових класів. Для масштабування до виробничих умов потрібні додаткові доменно-специфічні дані та повторне калібрування порога conf.

LLM-rate-limit. У разі пікового навантаження можливі відповіді 429/5xx. Резервним вирішенням може бути кеш embedding-ів або розгортання локального прискорювача (Raspberry Pi AI Kit) для зниження частки хмарних викликів.

Шумостійкість. При сильних артефактах JPEG локальна модель демонструє падіння F1 до 89 %. Можливе покращення через донавчання на розширеному наборі зашумлених даних чи використання більш сучасної lightweight-архітектури (наприклад, EfficientViT).

Приватність і аудит. Хоча сирі зображення не зберігаються у хмарі, журнал токенів і хешів може містити чутливі метадані; доцільно впровадити шифрування логів і ротацію ключів.

Висновки до розділу 5

Експериментальне дослідження показало, що запропонована гібридна система виконує вимоги, визначені у розділі 2. Комбінований режим із порогом упевненості 0,75 забезпечує F1 ≈ 97 %, середню затримку ≈ 105 мс та зменшує витрати на хмарні виклики більш ніж у 3 рази порівняно з повністю віддаленим обробленням, зберігаючи при цьому енергоспоживання Raspberry Pi у межах 5 Вт. Така

конфігурація виявилася найефективнішою для сценаріїв, що потребують низької латентності, прийнятної вартості та помірного апаратного ресурсу. Для переходу у промислову експлуатацію рекомендовано розширити датасет, додати механізм автоматичної ротації моделей та інтегрувати апаратний прискорювач, що підвищить стійкість і зменшить залежність від зовнішнього LLM-сервісу.

ВИСНОВКИ

Висновки по основній частині роботи

У роботі виконано повний цикл дослідження та реалізації гібридної системи класифікації об’єктів, що поєднує локальний inference на малопотужному пристрої з можливостями великої хмарної мультимодальної моделі. Отримані результати підтверджують актуальність і досяжність поставленої мети.

Теоретичний аналіз. Оцінено еволюцію класифікаційних нейромереж-від перцептрона до ViT і LLM-підходів. Виділено переваги та недоліки хмарних сервісів (висока точність, але велика латентність і залежність від каналу) та edge-моделей (низька затримка, але обмежена семантична гнучкість). Це обґрунтувало доцільність гібридної архітектури.

Обґрунтування напряму та методики. Сформульовано вимоги до системи: точність ≥ 95 %, альтентність edge-шляху ≤ 100 мс, мінімальне енергоспоживання, відсутність передачі сирих зображень у хмару. Вибрано поєднання EfficientNet-Lite INT8 (локально) та GPT-4o mini (віддалено) з автоматичною маршрутизацією за порогом упевненості.

Розробка архітектури. Реалізовано клієнт-серверну систему на Raspberry Pi 5: Nginx, FastAPI, gRPC-сервіс ai_worker. Забезпечено CSRF-, JWT-, CORS-захист, Tunnel Cloudflare без відкритих портів, автозапуск контейнерів і гаряче перемикання моделей через таблицю ai_models.

Алгоритмічна складова. Стандартизовано препроцес (224 × 224, JPEG ≤ 300 КБ), імплементовано INT8-квантизацію

та warm-up у ONNX Runtime. Алгоритм route_request() спрямовує запит у хмару лише за умови conf < 0,75 або перевищення розміру. Забезпечено шифрування TLS 1.3 і журнал дій для аудиту.

Експериментальна оцінка. На трьох вибірках (3 300 зображень) комбінований режим продемонстрував F1 = 96,9 %, медіанну затримку ≈ 105 мс та зменшив вартість хмарних викликів утричі порівняно з повністю віддаленим рішенням: 72 % кадрів оброблено локально, споживання Pi не перевищило 5 Вт.

Практична значущість. Отримана платформа придатна для швидкого розгортання у відеоспостереженні, промисловій інспекції та агромоніторингу, де важливі низька латентність і приватність даних. Архітектура легко масштабується (Swarm/K8s) та підтримує «канарейкове» оновлення моделей без простою.

Обмеження і перспективи. Потрібне розширення датасета для доменно-специфічних класів, впровадження кешування LLM-запитів і можливе додавання апаратного прискорювача (Raspberry Pi AI Kit) для подальшого зменшення частки хмарних звернень.

Таким чином, усі головні завдання, визначені у вступі, виконані: проведено теоретичний аналіз, розроблено й реалізовано гібридну систему, забезпечено вимоги до швидкодії та точності та підтверджено її ефективність експериментально.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*. 1958. Vol. 65, No 6. P. 386-408.

2. Rumelhart D.E., Hinton G.E., Williams R.J. Learning representations by back-propagating errors. *Nature*. 1986. Vol. 323, No 6088. P. 533–536.

3. LeCun Y., Bottou L., Bengio Y., Haffner P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. 1998. Vol. 86, No 11. P. 2278-2324.

4. Krizhevsky A., Sutskever I., Hinton G.E. ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems* 25. 2012. P. 1097-1105.

5. Dosovitskiy A. та ін. An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale // arXiv preprint arXiv:2010.11929. 2020.

6. Radford A. та ін. Learning transferable visual models from natural language supervision // arXiv preprint arXiv:2103.00020. 2021.

7. OpenAI. GPT-4o System Card. OpenAI, 2024. URL: https://openai.com/index/gpt-4o-system-card/ (дата звернення: 29.05.2025).

8. OpenAI. GPT-4o System Card. OpenAI, 2024. URL: https://openai.com/index/gpt-4o-system-card/ (дата звернення: 29.05.2025).

9. Google DeepMind. Introducing Gemini 2.0: our new AI model for the agentic era. – Google Blog, грудень 2024.

10. Anthropic. Claude 3 model family announcement. Anthropic Newsroom, 2024.

11. Howard A. та ін. Searching for MobileNet V3 // Proceedings of ICCV. 2019.

12. Tan M., Le Q. EfficientNet-Lite Models. Google AI Blog, 2020.

13. RangiLyu. NanoDet: Super lightweight real-time object detection on mobile devices. GitHub Repository, 2021.

14. ONNX Runtime Team. Quantizing an ONNX Model. onnxruntime.ai, 2024.

15. ONNX Runtime Team. Roadmap and Release 1.20.1. onnxruntime.ai, листопад 2024.

16. de Vries-Gao A. Energy consumption of artificial intelligence // Joule, 2025 ; огляд у WIRED, 23 травня 2025.

17. Rice T. Real Time Inference on Raspberry Pi 4. *PyTorch Tutorials*, 16 січня 2024.

18. EdgeIR. Edge AI vs. Cloud AI: Understanding the benefits and trade-offs of inferencing locations. – 16 квітня 2025.

19. Epoch AI. How much energy does ChatGPT use? – березень 2025.

20. Європейський Союз. Artificial Intelligence Act (OJ L 2024/1234 від 13 червня 2024).

21. Raspberry Pi Ltd. Benchmarking Raspberry Pi 5. – 12 листопада 2023.

22. ISACA. Understanding the EU AI Act. – Білий документ, 2024.

23. Zhang L. та ін. Optimizing Deep Learning Models for Raspberry Pi // arXiv:2304.13039, 2023.

24. EdgeIR. Edge AI vs. Cloud AI: Understanding the benefits and trade-offs of inferencing locations. – 16 квітня 2025.

25. NDIT Solutions. AI at the Edge: How real-time computing is redefining cloud strategy in 2025. – 12 лютого 2025.

26. Princeton Engineering. Leaner large language models could enable efficient local

use on phones and laptops. – 18 листопада 2024.

27. HPCwire. The Inference Bottleneck: Why Edge AI Is the Next Great Computing Challenge. – 15 квітня 2025.

28. ITPro Today. Hybrid Cloud and AI Emerge as Critical Priorities for IT Leaders in 2025. – 8 січня 2025.

29. Hassan B. Challenges and Future Directions: Energy Efficiency in Model Training and Inference // arXiv preprint arXiv:2404.13552. – 2024.

30. Junitec. Edge AI vs. Cloud AI: The Quiet Race to Cut Gen-AI's Energy Bill. – 6 травня 2025.

31. Xia H. та ін. Edge-assisted energy optimization for mobile AR applications // Scientific Reports. – 2025.

32. Thompson N. How Apple Intelligence's privacy stacks up against Android's hybrid AI // WIRED. – 12 липня 2024.

33. Business Insider. AI and 5G advancements will usher in the era of edge computing. – 28 лютого 2024.

34. Ткаліченко С. В. Штучні нейронні мережі: Навчальний посібник. Кривий Ріг: Державний університет економіки і технологій, 2023. 50 с.

35. Субботін С. О. Нейронні мережі : теорія та практика: навчальний посібник. Житомир : Вид. О. О. Євенок, 2020. 184 с.

36. Мозговенко А. А., Костромін К.Ю. Аналіз використання інструментів нейронних мереж при класифікації навчальних текстів дисциплін. *Матеріали II Всеукраїнської науково-практичної інтернет-конференції «Сучасні комп'ютерні та інформаційні системи і технології»*. Електронний ресурс. Режим доступу: http://www.tsatu.edu.ua/kn/wp-

content/uploads/sites/16/mozhovenko_144.pdf

ДОДАТКИ