Міністерство освіти і науки України
Національний технічний університет «Дніпровська політехніка»

Інститут електроенергетики

(навчально-науковий інститут)
Факультет інформаційних технологій

(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня магістр
(бакалавра, магістра)

Здобувача вищої освіти Гнатюк Олександр Васильович
(ПІБ)

академічної групи 123м-24-1
(шифр)

спеціальності 123 «Комп'ютерна інженерія»
(код і назва спеціальності)

спеціалізації за освітньо-професійною (освітньо-науковою) програмою
(за наявності)
« Комп'ютерна інженерія»

(офіційна назва)

на тему Інтелектуальна система моніторингу росту базиліку на гідропонній установці з використанням комп'ютерного зору
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	Булана Т.М.			
Рецензент				
Нормоконтролер				

Дніпро
2025

ЗАТВЕРДЖЕНО:

завідувач кафедри
інформаційних технологій та комп'ютерної інженерії
(повна назва)

_____ __В.В. Гнатушенко____
(підпис) (ініціали та прізвище)

«_____» _____20___року

ЗАВДАННЯ

на кваліфікаційну роботу ступеня_____магістр_____

(бакалавра, магістра)

здобувача вищої освіти_____Гнатюк О. В.____**академічної групи**___123м-24-1____
(прізвище та ініціали) (шифр)

спеціальності_____123 «Комп'ютерна інженерія»_____

спеціалізації за освітньою-професійною програмою «Комп'ютерна інженерія»
(за наявності)

на тему__ Інтелектуальна система моніторингу росту базиліку на гідропонній установці з використанням комп'ютерного зору_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від 13.10.2025 № 1165/с

Розділ	Зміст	Термін виконання
Розділ 1	Аналіз існуючих рішень детекції рослин в гідропонних умовах	2.10.2025-20.10.2020
Розділ 2	Моделі та методи при моніторингу росту базиліка на гідропонній установці	21.10.2025-2.11.2020
Розділ 3	Розробка системи моніторингу базиліку на гідропонній установці з використанням комп'ютерного зору	3.11.2025-22.11.2020
Розділ 4	Тестування на виробництві	23.11.2025-30.11.2020

Завдання видано _____ _____Булана Т.М._____
(підпис керівника) (ініціали та прізвище)

Дата видачі

Дата подання до екзаменаційної комісії_____

Прийнято до виконання _____ ___Гнатюк О. В.____
(підпис здобувача вищої освіти) (ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка: 65 стор., 5 рис., 1 додаток, 21 джерел.

Об’єктом дослідження є методи застосування комп’ютерного зору та нейронних мереж для аналізу зображень у процесах гідропонного вирощування рослин. Предметом дослідження є інтелектуальна система моніторингу росту базиліку на гідропонній установці з використанням алгоритмів комп’ютерного зору.

Метою магістерської роботи є розробка інтелектуальної інформаційної системи, яка забезпечує автоматизований контроль стану та динаміки росту базиліку в гідропонному середовищі на основі аналізу зображень та сучасних моделей нейронних мереж.

У вступі роботи обґрунтовано актуальність теми, що зумовлена зростанням попиту на автономні технології у точному рослинництві та розвитком гідропонних систем. Показано важливість застосування комп’ютерного зору для оптимізації процесів вирощування та підвищення ефективності контролю росту рослин. Окреслено основні завдання дослідження та проведено огляд сучасних технологій автоматизації аграрних і гідропонних установок.

Перший розділ присвячено аналізу існуючих рішень у сфері інтелектуального моніторингу рослин. Розглянуто особливості гідропонних систем, типи зображень, включно з RGB, мультиспектральними та інфрачервоними, та їхню роль у контролі вегетації. Описано джерела даних, які використовуються для моніторингу, зокрема стаціонарні камери. Проведено огляд методів класифікації, детекції та сегментації, що застосовуються для аналізу стану рослин, зокрема CNN, YOLO та Mask R-CNN.

Практична цінність роботи полягає у створенні інтелектуальної системи моніторингу росту базиліку в гідропонних умовах. Система дозволяє автоматизувати контроль росту, оптимізувати використання ресурсів та підвищити стабільність і якість вирощування. Розроблене рішення може бути

використане як у промислових гідропонних фермах, так і в малих домашніх установках.

Інформаційна система розроблена з використанням сучасного технологічного стеку. У якості бекенду застосовано Python та фреймворк FastAPI з сервером Uvicorn. Для роботи з базою даних використано SQLite, ORM SQLAlchemy та Alembic для управління міграціями. Валідація та серіалізація даних здійснюється за допомогою Pydantic. Для контейнеризації та спрощення розгортання використано Docker.

Фронтенд реалізовано на React 19 з TypeScript та бібліотекою MUI7 для побудови інтерфейсу, Axios та React-Query для обміну даними з сервером, CSS-Modules для стилізації компонентів. Nginx використовується для віддачі зібраних сторінок на Vue JS.

Ключові слова: КОМП’ЮТЕРНИЙ ЗІР, НЕЙРОННІ МЕРЕЖІ, YOLOV8, ГІДРОПОНІКА, МОНІТОРИНГ РОСЛИН, БАЗИЛІК, АВТОМАТИЗАЦІЯ, FASTAPI, REACT, DOCKER.

ЗМІСТ

Перелік умовних позначень

API (Application Programming Interface) — набір правил для взаємодії між програмними модулями або сервісами.

CNN (Convolutional Neural Network) — модель глибокого навчання для автоматичного виділення просторових ознак із зображень.

CPU (Central Processing Unit) — універсальний обчислювальний блок, який виконує основні логічні операції системи.

FPN (Feature Pyramid Network) — архітектура, що дозволяє ефективно використовувати ознаки різних масштабів.

GPU (Graphics Processing Unit) — апаратний компонент, що забезпечує паралельні обчислення для прискорення навчання нейронних мереж.

IoU (Intersection over Union) — показник оцінки точності детекції шляхом порівняння площ перекриття.

LabelImg (Label Image) — програмний засіб для створення та редагування анотацій зображень.

NMS (Non-Maximum Suppression) — алгоритм відбору найкращих передбачень шляхом усунення надлишкових рамок.

PANet (Path Aggregation Network) — мережа для покращення передачі інформації між рівнями ознак.

Pascal VOC (Visual Object Classes) — загальноприйнятий стандарт структури даних для задач комп’ютерного зору.

RNN (Recurrent Neural Network) — нейронна мережа для обробки послідовних даних із урахуванням попередніх станів.

SE (Squeeze-and-Excitation) — механізм підсилення інформативних каналів ознак у нейронній мережі.

SGD (Stochastic Gradient Descent) — метод оптимізації параметрів моделі під час навчання.

SSD (Single Shot MultiBox Detector) — метод детекції об’єктів, що виконує прогнозування за один прохід мережі.

ViT (Vision Transformer) — нейронна модель, що застосовує механізм уваги для аналізу візуальних даних.

XML (Extensible Markup Language) — формат збереження структурованих даних для опису анотацій.

YOLO (You Only Look Once) — підхід до детекції об’єктів у реальному часі на основі одного проходу моделі.

YOLOv8 (You Only Look Once, version 8) — сучасна реалізація YOLO з покращеною точністю та швидкістю роботи.

ВСТУП

Споконвіку сільське господарство посідало провідне місце серед ключових галузей людської діяльності, оскільки забезпечує населення продовольчими продуктами та сировиною для промисловості. Протягом розвитку цивілізації ця галузь зазнала значних трансформацій, спрямованих на підвищення ефективності виробництва, зменшення втрат і раціональне використання ресурсів. Сучасна епоха, позначена швидким розвитком інформаційних технологій і штучного інтелекту, створює нові можливості для цифровізації аграрного сектору.

Одним із найважливіших викликів сьогодні є боротьба з хворобами рослин. Вчасне виявлення патологій має ключове значення для збереження врожаю, однак традиційні методи діагностики часто вимагають значних людських зусиль, часу та спеціальних знань[2]. Використання технологій машинного навчання, зокрема нейронних мереж, відкриває нові шляхи для автоматизації цього процесу. Такі системи здатні аналізувати зображення рослин, визначати наявність уражень і з високою точністю локалізувати проблемні ділянки [3].

Метою даної роботи є дослідження та розробка інформаційної системи, яка забезпечує автоматизований аналіз стану рослин із застосуванням нейронних мереж. Зокрема, у межах дослідження передбачено:

- розглянути основні види хвороб рослин, їхні ознаки та фактори виникнення.

- проаналізувати сучасні системи вирощування рослин, зокрема агропоніку, гідропоніку, аеропоніку та інші варіанти контрольованого середовища.

- визначити методи, що застосовуються для підтримання оптимальних умов росту — системи моніторингу, автоматизоване керування мікрокліматом, освітленням і зрошенням.

Використання інструментів комп’ютерного зору, таких як YOLO, Faster R-CNN, U-Net та Vision Transformer, забезпечить високу точність і швидкість

обробки зображень, що є надзвичайно важливим для практичного застосування системи[4]. Запропоноване рішення може бути використане як у невеликих фермерських господарствах і теплицях, так і у великих агропромислових комплексах.

Для реалізації поставленої мети необхідно виконати такі завдання: проаналізувати існуючі методи обробки та інтерпретації зображень рослин із використанням нейронних мереж; сформувати структуру даних і підготувати навчальні вибірки для побудови та тренування моделей; розробити інформаційну систему, що реалізує алгоритми розпізнавання та класифікації стану рослин; провести експериментальне дослідження роботи системи, здійснити її тестування й оцінити показники точності та ефективності.

Впровадження інноваційних технологій у сільському господарстві створює нові можливості для підвищення продуктивності, сприяння сталому розвитку та зміцнення конкурентоспроможності галузі. Це дослідження зосереджене на розробці передових методів моніторингу здоров’я рослин та оптимізації сільськогосподарських процесів у умовах глобальних викликів

1 ОГЛЯД НАУКОВИХ І ПРИКЛАДНИХ ДЖЕРЕЛ

1.1 Сучасні тенденції моніторингу росту рослин у гідропонних системах

Гідропоніка є сучасною технологією вирощування рослин без ґрунту, у водних розчинах, збагачених поживними речовинами. Такий підхід дозволяє оптимізувати використання ресурсів, прискорити ріст культур та підвищити врожайність порівняно з традиційними методами [5, 10, 12]. Сучасні системи «розумного» сільського господарства активно інтегрують автоматизовані технології моніторингу та контролю, використовуючи сенсорні мережі, інструменти комп'ютерного зору та алгоритми штучного інтелекту (ШІ) [1–4].

Основна мета таких систем полягає у зменшенні впливу людського фактора, підвищенні стабільності врожайності та швидкому реагуванні на зміни стану рослин. Візуальна діагностика листкової поверхні рослин є ключовим інструментом, оскільки дозволяє відстежувати динаміку росту та виявляти ознаки дефіциту макро- та мікроелементів (азот, магній, фосфор) або прояви стресових факторів, таких як недостатнє освітлення, перегрів або патогенне ураження [1, 2, 3].

Сучасні дослідження показують, що використання алгоритмів глибокого навчання, зокрема CNN, YOLO та Vision Transformer, значно підвищує точність і швидкість аналізу зображень рослин [1, 2]. Зокрема, застосування моделей YOLOv8 дозволяє в реальному часі виявляти ознаки хвороб і дефіциту живлення, забезпечуючи ефективне управління гідропонними установками [4, 17, 20]. Інтеграція цих технологій із сенсорними мережами та IoT-пристроями дозволяє отримувати багатомасштабні дані про стан рослин, включаючи інформацію про температуру, вологість, освітлення та рівень поживних речовин у розчині [5].

Крім того, сучасні системи моніторингу не обмежуються лише виявленням захворювань. Вони дозволяють прогнозувати темпи росту, оптимізувати режим освітлення та зрошення, а також автоматично коригувати параметри середовища для забезпечення стабільного розвитку рослин [5, 6, 10, 12, 14]. У роботах Nyawose (2025) та Pascal (2024) підкреслюється важливість інтеграції глибокого

навчання та обробки зображень для створення адаптивних систем, які здатні враховувати індивідуальні особливості культур та змінні умови середовища [2].

Особливу увагу в останніх дослідженнях приділено розробці інтелектуальних гідропонних систем для контролю росту пряно-ароматичних рослин, зокрема базиліку. Застосування мультимодальних підходів, що об’єднують RGB-зображення, дані сенсорів і алгоритми машинного навчання, дозволяє отримувати високоточну інформацію про стан листя, кореневої системи та загальний розвиток рослин [9, 10, 18]. Це забезпечує ефективне планування живлення та поливу, а також дозволяє прогнозувати потенційний урожай і своєчасно реагувати на негативні фактори.

Таким чином, сучасні тенденції розвитку інтелектуальних систем моніторингу гідропонних культур демонструють високу ефективність поєднання комп’ютерного зору, машинного навчання та IoT-технологій. Використання цих підходів дозволяє створювати автоматизовані рішення, здатні забезпечувати точний, своєчасний та адаптивний контроль за ростом рослин у контрольованому середовищі, підвищуючи продуктивність і стабільність вирощування [15–18].

1.2 Апаратні засоби збору даних

Для ефективного функціонування інтелектуальних систем моніторингу гідропонних культур критично важливим є використання сучасних апаратних засобів збору даних. Вони забезпечують надійну основу для збору візуальної та сенсорної інформації, необхідної для аналізу стану рослин і прийняття автоматизованих рішень.

Одним із базових компонентів є цифрові камери високої роздільності (HD, 4K), які дозволяють отримувати детальні зображення листкової поверхні рослин. Такі камери можуть бути встановлені стаціонарно над грядками або на рухомих рейках, що забезпечує можливість регулярного і систематичного збору даних [6, 9, 10]. Використання високоякісних камер дозволяє точно оцінювати розміри

листя, їх колір, структуру та наявність видимих дефектів, що є критично важливим для раннього виявлення дефіциту елементів живлення та хвороб.

Для більш глибокого аналізу стану рослин застосовуються інфрачервоні (IR) та мультиспектральні камери, які дозволяють оцінювати індекси вегетації, зокрема NDVI (Normalized Difference Vegetation Index) та інші спектральні показники. Ці індекси корелюють з рівнем фотосинтетичної активності та загальним здоров'ям рослин, що дозволяє виявляти стресові фактори ще до появи видимих симптомів [14]. Мультиспектральні дані широко застосовуються у сучасних дослідженнях інтелектуальних гідропонних систем, оскільки вони дозволяють інтегрувати інформацію про біофізичні характеристики рослин у алгоритми машинного навчання .

У малих лабораторних та навчальних експериментах ефективними є RGB-камери смартфонів або компактні веб-камери. Вони дозволяють швидко створювати бази даних зображень рослин для тренування та тестування моделей глибокого навчання, що робить дослідження більш доступним і гнучким [2, 3, 7]. Сучасні системи моніторингу обов'язково включають сенсорні модулі Internet of Things (IoT), які контролюють параметри мікроклімату: температуру, вологість, рівень CO_2, освітленість та концентрацію розчинених речовин у поживному розчині. Дані сенсорів інтегруються з візуальною інформацією і слугують додатковими вхідними параметрами для алгоритмів машинного навчання та нейронних мереж, що дозволяє моделювати динаміку росту рослин та прогнозувати їхній стан [15].

Особливо перспективним є використання розумних камер на базі платформ NVIDIA Jetson, Raspberry Pi + PiCam, ESP32-CAM та аналогічних рішень. Вони поєднують апаратну та програмну частину, здатні проводити первинну обробку зображень безпосередньо на місці збору даних (edge computing). Це дозволяє зменшити затримки при передачі великих обсягів даних до центральної серверної системи та забезпечує оперативну реакцію на зміни стану рослин [18].

Таким чином, апаратна частина інтелектуальної системи моніторингу може бути побудована як комплекс взаємопов’язаних камер та сенсорів, інтегрованих через локальну мережу або бездротові канали передачі даних до серверної частини для подальшої обробки та аналізу. Такий підхід забезпечує комплексний збір інформації про стан рослин, підвищує точність діагностики та дозволяє реалізовувати автоматизоване управління умовами вирощування у реальному часі [9].

1.3 Методи та алгоритми аналізу зображень

Аналіз зображень є ключовим етапом у процесі моніторингу росту рослин у гідропонних системах. Використання методів комп’ютерного зору дозволяє автоматизувати оцінку візуальних характеристик рослин, зменшити вплив людського фактора та підвищити точність виявлення ознак стресу або хвороб. У сучасних дослідженнях застосовуються як класичні методи обробки зображень, так і алгоритми глибинного навчання, що забезпечують більш високий рівень точності та адаптивності [1].

Класичні методи обробки зображень включають сегментацію за кольором у просторі HSV, виділення контурів за допомогою операторів Canny та Sobel, а також морфологічні операції, такі як ерозія, дилатація та відкриття/закриття. Такі методи дозволяють проводити первинний аналіз листкової поверхні, оцінювати площу листя та визначати наявність видимих пошкоджень [2, 3, 19]. Одним із напрямів є виділення ознак стресу рослин, що включає аналіз інтенсивності зеленого кольору, оцінку хлорозу, некрозу або деформації листкової пластини на основі змін текстури. Це дає змогу своєчасно виявляти дефіцит поживних речовин, недостатнє освітлення, перегрів або ураження патогенами [1, 3, 5, 7].

Сучасні підходи ґрунтуються на методах глибинного навчання, що дозволяють значно підвищити точність аналізу та автоматизувати процес класифікації і детекції. Convolutional Neural Networks (CNN), включаючи архітектури ResNet, MobileNet та EfficientNet, широко застосовуються для класифікації стану рослин, розпізнавання хвороб та виявлення дефектів . Для сегментації листкової поверхні та оцінки площі листя використовуються U-Net

та Mask R-CNN, що дозволяє створювати точні маски листкових областей навіть у складних умовах освітлення або при частковому перекриванні листя [13].

Методи детекції об’єктів, такі як YOLOv8 та Detectron2, застосовуються для визначення кількості листків, їх розташування та виявлення конкретних зон ураження рослин. Висока швидкість роботи цих моделей дозволяє реалізовувати моніторинг у реальному часі та інтегрувати його в системи автоматичного керування гідропонними установками [17, 20].

Особливе значення має аналіз росту рослин у часі, що реалізується за допомогою рекурентних нейронних мереж (RNN, LSTM) або гібридних моделей CNN+LSTM. Ці методи дозволяють відстежувати динаміку зміни площі листя, прогнозувати швидкість росту та оцінювати ефективність різних режимів освітлення, зрошення та живлення [10].

Сучасні дослідження показують, що інтеграція даних з RGB-камер, мультиспектральних сенсорів та параметрів мікроклімату значно підвищує точність моделей машинного навчання. Використання багатомодальних даних дозволяє будувати більш надійні прогностичні алгоритми, що враховують не лише візуальні характеристики, а й фізіологічний стан рослин, умови освітлення та рівень вологості [15].

Таким чином, поєднання класичних методів обробки зображень із сучасними алгоритмами глибинного навчання та багатомодальним аналізом даних забезпечує високу ефективність інтелектуальних систем моніторингу гідропонних культур, зокрема для вирощування базиліку, та дозволяє реалізувати автоматизовану оцінку стану рослин у реальному часі [20].

1.4 Сховища даних та хмарні технології

У сучасних інтелектуальних системах моніторингу рослин особливу роль відіграють ефективні механізми зберігання та обробки даних. Через велику кількість інформації, що надходить із камер високої роздільної здатності, сенсорів IoT та інших джерел, необхідно забезпечити надійне збереження даних, їхню масштабованість, швидкий доступ і безпеку. Для цього застосовуються

хмарні платформи, які надають комплексні сервіси для обробки, аналітики та інтеграції з машинним навчанням [5, 8, 12, 18].

Серед найбільш поширених рішень використовують Google Cloud Platform (GCP), яка забезпечує зберігання зображень у Google Cloud Storage і розгортання моделей машинного навчання через сервіс Vertex AI. Платформа підтримує інтеграцію з різними аналітичними та обчислювальними модулями, що дозволяє будувати масштабовані системи моніторингу [5, 9].

Також широко застосовуються сервіси Amazon Web Services (AWS), зокрема модулі Rekognition для автоматичного виявлення об’єктів та SageMaker для розробки, тренування та розгортання моделей машинного навчання. Це дозволяє здійснювати підготовку даних, налаштування гіперпараметрів та автоматизацію процесу навчання моделей у продуктивному середовищі [10].

Microsoft Azure AI надає можливості для інтеграції IoT та комп’ютерного зору у сільському господарстві. Сервіси аналітики зображень через Computer Vision API та IoT Hub дозволяють централізовано керувати сенсорами, моніторити стан обладнання та збирати дані про мікроклімат у реальному часі .

Для зберігання структурованих даних і метаданих застосовуються Firebase, Supabase та MongoDB Atlas. Вони дозволяють будувати веб-інтерфейси реального часу, відображати динаміку росту рослин та інтегруватися з фронтенд-додатками на React або Vue. Ці сервіси підтримують масштабування баз даних без втрати продуктивності та забезпечують безпечний доступ через аутентифікацію і шифрування [5, 9, 11].

Завдяки контейнеризації з використанням Docker та Kubernetes забезпечується гнучке розгортання аналітичних модулів, автоматичне масштабування сервісів та інтеграція з веб-платформами. Це дозволяє виконувати складні обчислення безпосередньо у хмарі або на локальних серверах, підтримуючи високу доступність системи та можливість підключення нових сенсорів і камер у будь-який момент [5, 8, 12, 18].

Таким чином, використання хмарних технологій і сучасних баз даних дозволяє створювати ефективні, масштабовані та безпечні інтелектуальні

системи моніторингу росту рослин, включно з гідропонними установками, забезпечуючи безперервний збір, обробку та аналіз великих обсягів даних у реальному часі. Поєднання IoT, комп'ютерного зору та хмарних платформ створює основу для побудови високотехнологічного та адаптивного середовища для контролю росту та стану рослин [5–6, 8–12, 18].

1.5 Інтелектуальні системи моніторингу рослин

Сучасний ринок пропонує широкий спектр інтелектуальних систем та платформ для моніторингу стану рослин, що дозволяють автоматизувати процеси контролю росту, виявлення хвороб та оптимізації агротехнічних умов. Серед мобільних застосунків, які здобули популярність серед фермерів та дослідників, можна виділити Plantix, LeafDoc та AgroAI. Ці програми здатні за допомогою аналізу зображень розпізнавати ознаки ураження рослин, класифікувати тип хвороби та надавати рекомендації щодо лікування [11, 12, 5]. Їхня перевага полягає у простоті використання та швидкому отриманні результатів, проте вони зазвичай універсальні і не оптимізовані під конкретні культури, такі як базилік.

Апаратно-програмні комплекси, такі як FarmBot, GroLab та HydroGrow, поєднують сенсорні технології, робототехніку та алгоритми машинного навчання для автоматизації гідропонних систем. Вони дозволяють здійснювати контроль температури, освітлення, вологості, автоматичне поливання та живлення рослин. Додатково використовується комп'ютерне зору для оцінки розвитку рослин у реальному часі, що дозволяє виявляти дефекти росту, хлороз, некроз або інші ознаки стресу [5, 6, 10].

Відкриті платформи, такі як OpenAg, AgriSense та eAgronom, забезпечують гнучкість у побудові систем моніторингу. Вони дозволяють користувачу підключати власні камери, сенсори IoT, формувати бази даних зображень та тренувати нейронні мережі для специфічних завдань. Це відкриває можливість створення персоналізованих рішень, адаптованих під конкретні культури та умови вирощування [8, 9, 11].

Проте більшість існуючих комерційних та відкритих систем мають універсальний характер і не враховують особливості окремих рослин. Наприклад, для базиліку важливі специфічні параметри освітлення, поливу та температурного режиму, а також особливості листкової поверхні для точного визначення динаміки росту та ознак стресу. Саме тому розробка спеціалізованої інтелектуальної системи для моніторингу росту базиліку на гідропонній установці є актуальною і науково обґрунтованою задачею.

Такі системи повинні поєднувати апаратні засоби збору даних (камери, мультиспектральні сенсори, IoT-модулі), алгоритми комп'ютерного зору та глибинного навчання для обробки великих масивів інформації, а також хмарні або локальні сховища для масштабованого зберігання та аналізу даних. Впровадження таких рішень дозволяє забезпечити автоматизований моніторинг росту рослин у реальному часі, підвищити ефективність використання ресурсів, знизити людський фактор і поліпшити стабільність та якість врожаю [5–6].

Таким чином, інтеграція інтелектуальних технологій у системи гідропонного вирощування рослин створює передумови для побудови спеціалізованих рішень, адаптованих під конкретні культури, забезпечує підвищення точності моніторингу та оптимізацію агротехнічних процесів, що робить цю тему надзвичайно актуальною для сучасного сільського господарства та наукових досліджень.

1.6 Висновки до розділу

Проведений аналіз сучасних технологій та наукових досліджень у сфері інтелектуальних систем моніторингу рослин дозволяє виділити кілька ключових висновків. Сучасні ефективні системи моніторингу формуються на основі комплексної інтеграції різних компонентів. Серед них важливе місце займає апаратна частина, до якої входять цифрові та мультиспектральні камери, сенсорні модулі Internet of Things (IoT) та мікроконтролери, що забезпечують збір даних і первинну обробку інформації безпосередньо на місці вирощування.

Не менш значущою є програмна складова, представлена алгоритмами комп'ютерного зору, які дозволяють здійснювати сегментацію листкової поверхні, класифікацію стану рослин, оцінку площі листків та аналіз ознак стресу, дефіциту поживних речовин або впливу інших факторів середовища. До цього додаються моделі машинного та глибинного навчання, зокрема CNN, YOLOv8, U-Net, RNN/LSTM, що забезпечують прогнозування динаміки росту рослин, автоматичне виявлення хвороб та оцінку ефективності агротехнічних заходів.

Для зберігання великих обсягів даних і забезпечення їхньої безпеки, масштабованості та доступності для аналітики використовуються хмарні або локальні платформи, які також дозволяють організовувати веб-інтерфейси для моніторингу стану рослин у режимі реального часу [5, 6, 8–12, 18].

Водночас, аналіз існуючих рішень показав, що більшість комерційних систем є універсальними та не враховують специфіку окремих культур. Це обумовлює потребу у створенні спеціалізованих інтелектуальних систем, адаптованих до конкретних рослин, умов їх вирощування та локальних параметрів гідропонних установок.

У зв'язку з цим, актуальним завданням залишається розробка адаптивної системи моніторингу росту базиліку із застосуванням сучасних методів комп'ютерного зору та машинного навчання. Розроблювана в межах дипломного проєкту система має забезпечити автоматизований контроль стану рослин, прогнозування їх динаміки росту та оперативне виявлення ознак стресу або дефіциту поживних речовин у режимі реального часу, що дозволить підвищити ефективність, стабільність та продуктивність процесів вирощування базиліку на гідропонній установці.

1.7 Постановка задачі

У результаті аналізу сучасних інтелектуальних систем моніторингу росту рослин та огляду наявних наукових досліджень встановлено, що більшість існуючих наборів даних орієнтовані на універсальні культури, такі як пшениця, кукурудза, томати чи салат, і не враховують специфіку вирощування рослин у

гідропонних системах. На момент проведення дослідження відсутній відкритий, структурований та спеціалізований датасет, спрямований саме на моніторинг росту базиліку в умовах гідропоніки.

Відсутність такого набору даних ускладнює процес навчання і валідації моделей комп'ютерного зору, знижує точність оцінки росту рослин і обмежує можливості порівняння результатів між різними дослідженнями. Це створює потребу у формуванні власного авторського датасету, який був би адаптований до умов конкретної гідропонної установки та особливостей розвитку базиліку.

У межах даного дипломного проєкту передбачено розробку та формування такого спеціалізованого набору даних. Він включатиме послідовності RGB-зображень рослин базиліку, зроблених на різних етапах росту, а також метадані умов вирощування, що відображають температуру, вологість, освітленість і часові мітки. Крім того, датасет міститиме анотації зображень для виконання завдань сегментації листкової поверхні, оцінки площі листків та аналізу динаміки росту.

На основі цього датасету планується реалізувати декілька ключових завдань. По-перше, розробити та навчити моделі комп'ютерного зору для автоматизованого моніторингу росту базиліку. По-друге, оцінити ефективність різних алгоритмів обробки та аналізу зображень. По-третє, сформувати рекомендації щодо подальшого використання створеного датасету у наукових та прикладних дослідженнях.

Таким чином, основною метою цієї частини дипломної роботи є створення спеціалізованого датасету базиліку для гідропонних систем, який стане основою для побудови інтелектуальної системи моніторингу росту рослин із застосуванням сучасних методів комп'ютерного зору. Це дозволить підвищити точність оцінки стану рослин, автоматизувати процес моніторингу та забезпечити надійну базу для подальших наукових досліджень у сфері інтелектуального рослинництва.

2. МОДЕЛІ ТА МЕТОДИ ПРИ МОНІТОРИНГУ РОСТУ БАЗИЛІКА НА ГІДРОПОННІЙ УСТАНОВЦІ

2.1 Формалізація візуальних даних у системі моніторингу росту рослин

Розробка інтелектуальної системи моніторингу росту базиліку на гідропонній установці передбачає чітку формалізацію та структуризацію візуальних даних, що надходять із камер спостереження. У цьому контексті комп'ютерний зір є основним джерелом інформації про фізичний стан рослин, їхню динаміку росту, морфологічні зміни листя та загальну біомасу.

Візуальні дані мають багаторівневу структуру, яка включає як низькорівневі числові характеристики, так і високорівневі семантичні ознаки. Така ієрархія дозволяє поєднувати локальні особливості, наприклад форму та текстуру листків, із глобальними показниками росту і стану рослини в цілому. Це забезпечує точний та комплексний аналіз розвитку рослин у часі, що є критично важливим для автоматизованого моніторингу та прогнозування їх стану [14, 15].

2.1.1 Первинне подання зображень у вигляді піксельних ознак

На базовому рівні вся інформація, що надходить до системи комп'ютерного зору, представлена у вигляді пікселів. Піксель є мінімальною одиницею цифрового зображення та характеризується набором числових значень, які відповідають інтенсивності світлового сигналу.

Для кольорових зображень, що використовуються при спостереженні за ростом базиліку, застосовується модель RGB, у якій кожен піксель описується трьома каналами: червоним, зеленим та синім. Значення кожного каналу зазвичай знаходяться в діапазоні від 0 до 255, що дозволяє точно передавати колірні відтінки листя, стебел та субстрату.

Окрім стандартних RGB-зображень, у системах агромоніторингу можуть застосовуватись монохромні або інфрачервоні зображення. Монохромні дані спрощують обробку у випадках, коли колір не є визначальною ознакою, а інфрачервоні канали дозволяють опосередковано оцінювати фізіологічний стан рослин, наприклад рівень випаровування або температурні аномалії.

2.1.2 Виділення морфологічних елементів рослини

Наступним рівнем абстракції є формування візуальних об’єктів, що мають біологічне значення в контексті росту базиліку. До таких об’єктів належать листки, стебла, точки росту та контури всієї рослини.

Завдання інтелектуальної системи полягає не лише у визначенні присутності рослини на зображенні, а й у виділенні її структурних компонентів. Це дозволяє оцінювати площу листкової поверхні, зміни форми листя, щільність розташування рослин та інші важливі агрономічні показники, що відображають стан розвитку культури.

Для опису просторового розташування морфологічних елементів застосовуються геометричні примітиви, такі як обмежувальні прямокутники (bounding boxes) або піксельні маски (pixel masks). Таке подання формує основу для подальших етапів аналізу, зокрема для кількісної оцінки росту, відстеження динаміки розвитку та порівняння стану рослин у різні часові проміжки.

2.1.3 Матричне та тензорне представлення даних

З математичної точки зору, зображення, що надходять у систему, представлені у вигляді багатовимірних масивів даних. Для кольорових зображень формується тривимірний тензор, який включає просторові координати пікселів та канали кольору (RGB). Таке представлення є стандартним підходом у сучасних методах глибокого навчання та комп’ютерного зору.

У процесі обробки ці тензори зазнають різних трансформацій: вони можуть змінювати розмір (resizing), комбінуватися або перетворюватися у багатовимірні ознакові простори. Саме в такому вигляді дані передаються між шарами нейронних мереж, що дозволяє поступово переходити від сирих піксельних значень до узагальнених ознак, які відображають стан росту та морфологічні характеристики базиліку. Це забезпечує ефективне виявлення структурних змін рослин і дозволяє системі здійснювати точний аналіз динаміки їхнього розвитку.

2.1.4 Ієрархічна модель візуальних ознак росту

У задачах моніторингу росту базиліку багаторівнева структура візуальних ознак дозволяє ефективно поєднувати локальні зміни з глобальними показниками розвитку рослини. Нижній рівень включає піксельні характеристики, що відображають зміни кольору, текстури та освітленості листкової поверхні. На цьому рівні застосовуються класичні методи обробки зображень, такі як сегментація в HSV-просторі, детектори контурів (Canny, Sobel), морфологічні операції для очищення шуму та виділення листкової поверхні.

Середній рівень охоплює морфологічні елементи, такі як листки, частини стебел та точки росту. Для їхньої обробки застосовуються сучасні моделі глибинного навчання, зокрема Convolutional Neural Networks (CNN) та їх модифікації: ResNet, EfficientNet, MobileNet. CNN дозволяють виділяти локальні ознаки та класифікувати стан листя, наприклад, на здорове, уражене хворобою або стресоване. Для завдань сегментації використовуються U-Net та Mask R-CNN, що дозволяє виділяти піксельні маски окремих листків та оцінювати площу листкової поверхні. Для детекції об’єктів та оцінки кількості листків застосовуються YOLOv8 або Detectron2, які забезпечують швидке та точне виявлення навіть при високій щільності рослин на зображенні.

Вищий рівень формують інтегральні показники, що описують стан рослини в цілому. Для їхнього формування використовується об’єднання локальних даних із середнього рівня та часових рядів зображень. Тут застосовуються рекурентні нейронні мережі (RNN, LSTM) або гібридні моделі CNN+LSTM, які дозволяють аналізувати динаміку росту, оцінювати швидкість розвитку листків та передбачати можливі проблеми у розвитку рослини. Таке поєднання забезпечує не тільки відстеження поточного стану базиліку, а й прогнозування його розвитку у режимі реального часу, що є критично важливим для автоматизованого управління умовами гідропонної установки.

Технічно дані з камер подаються у вигляді багатовимірних тензорів. Для кольорових зображень формується тривимірний тензор, де два виміри

відповідають просторовим координатам, а третій — каналам кольору (RGB). Після цього дані можуть трансформуватися через шари нейронної мережі: масштабуватися, нормалізуватися, комбінуватися з іншими сенсорними даними (температура, вологість, освітленість) та перетворюватися у багатовимірні ознакові простори. Це дозволяє поступово переходити від сирих піксельних значень до абстрактних характеристик, таких як форма листка, площа листкової поверхні або загальна густота рослин на грядці.

У роботі застосовується підхід, який поєднує три рівні аналізу: піксельний, морфологічний і інтегральний. На піксельному рівні відбувається первинна обробка зображень та виділення локальних ознак. На морфологічному рівні відбувається сегментація та класифікація структур рослини, а на інтегральному рівні — обчислення агрегованих показників росту та динаміки розвитку. Така ієрархія дозволяє інтелектуальній системі забезпечувати точний моніторинг росту базиліку, автоматично виявляти ознаки стресу та дефіциту поживних речовин, прогнозувати розвиток рослини та надавати рекомендації щодо оптимізації умов гідропонного вирощування.

Таким чином, поєднання багаторівневого аналізу візуальних ознак, сучасних моделей глибинного навчання та інтеграції сенсорних даних створює ефективну основу для побудови інтелектуальної системи моніторингу росту базиліку. Такий підхід дозволяє підвищити точність оцінки стану рослин, скоротити час реагування на зміну умов вирощування та забезпечити стабільне і якісне вирощування у гідропонній установці.

2.2 Моделі глибокого навчання для аналізу росту базиліку

Моделі глибокого навчання є ключовим інструментом для автоматизованого аналізу росту базиліку на гідропонній установці. Вони дозволяють ефективно обробляти великі обсяги візуальних даних та виділяти складні закономірності, що важко помітні при традиційному аналізі. Для задач класифікації стану рослин та виявлення ознак стресу застосовуються згорткові нейронні мережі (CNN), які здатні виділяти локальні ознаки листкової поверхні,

оцінювати її площу та визначати наявність пошкоджень. Моделі сегментації, такі як U-Net та Mask R-CNN, дозволяють точно виділяти контури листків та стебел, що необхідно для кількісної оцінки росту та щільності рослин. Для детекції та підрахунку листків у реальному часі застосовуються алгоритми типу YOLOv8, які поєднують швидкість роботи та високу точність. Крім того, для аналізу динаміки росту у часі використовуються рекурентні нейронні мережі (RNN) та моделі LSTM, що дозволяють відстежувати розвиток рослин протягом експерименту та прогнозувати майбутні зміни. Інтеграція цих моделей у систему моніторингу забезпечує комплексний підхід до оцінки стану базиліку та автоматизації управління умовами гідропонного вирощування.

2.2.1 Згорткові нейронні мережі для вилучення візуальних ознак

Згорткові нейронні мережі (CNN) є основою для обробки зображень у системах комп’ютерного зору завдяки здатності автоматично виділяти просторові ознаки з візуальних даних. Ця особливість робить їх надзвичайно ефективними для задач, пов’язаних із аналізом структури рослин, зокрема оцінки стану листя та морфологічних змін. Перші архітектури, такі як LeNet, заклали базові принципи використання згорткових шарів для класифікації зображень, а подальший розвиток привів до створення більш глибоких і складних моделей, таких як AlexNet, VGG та ResNet. Ці мережі здатні обробляти великі обсяги даних та працювати зі складними зображеннями з високою точністю. У контексті моніторингу росту базиліку CNN дозволяють формувати детальні ознакові описи рослини, включаючи форму листя, текстуру поверхні, інтенсивність забарвлення та інші показники, що відображають динаміку росту та вплив стресових факторів, таких як дефіцит поживних речовин або зміни умов освітлення та вологості. Такий підхід забезпечує точний і автоматизований аналіз розвитку базиліку на всіх етапах вирощування.

2.2.2 Моделі детекції об’єктів для аналізу росту та уражень базиліку

У задачі інтелектуального моніторингу росту базиліку детекція об’єктів має ключове значення, оскільки саме на цьому етапі відбувається локалізація

рослин, визначення окремих зон ураження та виявлення аномальних ділянок листя [13]. Система використовує статичні RGB-зображення, отримані з камери ESP32-S3 із роздільною здатністю 1920×1080 пікселів і частотою зйомки раз на 20 хвилин. Такий режим не потребує обробки даних у реальному часі, проте висуває підвищені вимоги до точності локалізації та стабільності результатів при тривалому моніторингу.

Детектори об’єктів у цій системі виконують кілька основних функцій: визначають просторове положення рослини, виявляють локальні уражені ділянки листя, формують вихідні дані для подальшого аналізу росту в динаміці часу та дозволяють порівнювати стан рослин, що вирощуються в різних умовах. Серед найбільш поширених підходів до детекції об’єктів у комп’ютерному зорі для аграрних застосувань виділяють Faster R-CNN, SSD та YOLO, кожен з яких має свої переваги та обмеження [16]. Порівняльні характеристики моделей наведені в таблиці 2.1:

Таблиця 2.1 – Порівняння моделей детекції об’єктів для системи моніторингу росту базиліку

Модель детекції	**Тип архітектури**	**Точність локалізації**	**Швидкодія**	**Вимоги до ресурсів**	**Придатність для ESP32-S3**	**Доцільність використання**
Faster R-CNN	Двоетапна	Висока	Низька	Високі	Низька	Доцільна для офлайн-аналізу зображень високої якості
SSD	Одноетапна	Середня	Висока	Середні	Середня	Компроміс між швидкістю та точністю
YOLOv8	Одноетапна	Висока	Висока	Помірні	Висока	Оптимальний вибір для детекції рослин і зон ураження

Аналіз показує, що YOLOv8 є найбільш придатною моделлю для системи моніторингу росту базиліку. Вона поєднує високу точність локалізації з помірними вимогами до обчислювальних ресурсів і добре масштабується для

обробки статичних зображень високої роздільної здатності. Крім того, YOLOv8 підтримує навчання на власних датасетах, що є критично важливим для виявлення специфічних типів уражень базиліку [17, 20]. Такий підхід дозволяє не лише ефективно визначати положення рослин, але й оцінювати стан листків, контролювати динаміку росту та своєчасно реагувати на прояви стресу або дефіциту поживних речовин, що забезпечує стабільність та підвищує ефективність гідропонного вирощування. .

2.2.3 Сегментаційні моделі для оцінки площі листя та зон ураження

Для більш детального аналізу росту рослин та оцінки ступеня ураження однієї лише детекції об’єктів недостатньо. У таких випадках використовуються сегментаційні методи, які дозволяють класифікувати кожен піксель зображення та точно визначати контури листя та патологічних зон [19]. Завдяки цьому можна не лише локалізувати рослину, а й отримати кількісні показники стану її тканин, що критично важливо для оцінки росту, виявлення дефіциту поживних речовин або стресових факторів.

У системі моніторингу росту базиліку сегментація використовується для кількох основних задач:

- обчислення площі листкової поверхні, що дозволяє оцінити загальну біомасу рослини та швидкість її росту;
- визначення співвідношення здорових і уражених ділянок листя, що дає змогу своєчасно виявляти ознаки хвороб або фізіологічного стресу;
- аналіз зміни морфології рослини у часовій динаміці, включаючи зміну форми листків, щільність розміщення та розмір точок росту;
- отримання кількісних даних для порівняння розвитку рослин, вирощених у різних умовах освітлення, вологості чи поживного розчину.

Найбільш поширеними архітектурами для сегментації є U-Net та Mask R-CNN, які мають різні підходи до обробки зображень та різні вимоги до обчислювальних ресурсів. Порівняльні характеристики наведено в таблиці 2.2:

Таблиця 2.2 – Порівняння сегментаційних нейронних мереж для аналізу росту базиліку

Модель сегментації	**Тип сегментації**	**Точність контурів**	**Обчислювальна складність**	**Переваги**	**Недоліки**	**Доцільність використання**
U-Net	Семантична	Висока	Середня	Простота архітектури, точна сегментація листя	Не розрізняє окремі об’єкти	Оптимальна для оцінки площі листя
Mask R-CNN	Інстансна	Висока	Висока	Поєднання детекції та сегментації	Високі вимоги до ресурсів	Доцільна для складних сцен з кількома рослинами

U-Net є ефективним вибором для задач, де необхідно сегментувати листкову масу як єдину область, що особливо корисно для оцінки загальної біомаси та швидкості росту. Mask R-CNN, у свою чергу, дозволяє виконувати інстансну сегментацію, що робить можливим аналіз кожної рослини окремо, проте потребує значно більших обчислювальних ресурсів[21].

2.3 Висновки до розділу

У результаті детального аналізу сучасних методів і моделей комп’ютерного зору можна зробити висновок, що для ефективного інтелектуального моніторингу росту базиліку на гідропонній установці доцільно поєднувати моделі детекції та сегментації. Такий підхід дозволяє не лише локалізувати рослини на зображеннях, а й отримувати точні кількісні та якісні показники їх розвитку. З урахуванням технічних характеристик апаратної платформи ESP32-S3, специфіки режиму зйомки (статичні RGB-зображення високої роздільної здатності з інтервалом 20 хвилин) та поставлених завдань, оптимальним рішенням є використання детектора об’єктів YOLOv8 у поєднанні з сегментаційними нейронними мережами, такими як U-Net або Mask R-CNN.

Детектор YOLOv8 забезпечує швидку і точну локалізацію рослин та окремих зон ураження, що дозволяє системі оперативно ідентифікувати проблемні ділянки листкової поверхні та контролювати просторове положення рослин у межах гідропонної установки. Сегментаційні моделі, у свою чергу, дозволяють проводити піксельну класифікацію, точно визначати контури листя, окремі фрагменти стебел, точки росту та патологічні зони. Використання U-Net забезпечує ефективну семантичну сегментацію для оцінки загальної площі листкової поверхні та швидкості росту, а Mask R-CNN дозволяє проводити інстансну сегментацію, що дає змогу аналізувати кожну рослину окремо та відстежувати зміни морфології у складних багаторослинних сценах.

Поєднання цих моделей створює комплексну аналітичну систему, яка дозволяє отримувати кількісні та якісні показники росту базиліку, оцінювати рівень ураження листя, порівнювати розвиток рослин у різних умовах вирощування та формувати рекомендації щодо оптимізації агротехнічних заходів. Такий підхід формує надійну методологічну основу для подальшого проєктування архітектури інтелектуальної системи моніторингу, що передбачає інтеграцію апаратних сенсорів, алгоритмів комп’ютерного зору та машинного навчання для забезпечення високої точності, стабільності та масштабованості роботи системи в умовах реального часу.

3. РОЗРОБКА СИСТЕМИ МОНІТОРИНГУ БАЗИЛІКУ НА ГІДРОПОННІЙ УСТАНОВЦІ З ВИКОРИСТАННЯМ КОМП'ЮТЕРНОГО ЗОРУ

Розробка системи моніторингу росту базиліку на гідропонній установці з використанням комп'ютерного зору передбачає створення інтегрованої платформи, яка поєднує апаратні засоби збору даних, алгоритми аналізу зображень та інтелектуальні моделі для оцінки стану рослин. Основою системи є камера високої роздільної здатності, яка періодично фіксує зображення базиліку, та сенсорні модулі для контролю параметрів середовища — температури, вологості, освітленості та складу поживного розчину. Комп'ютерний зір забезпечує автоматичну обробку цих зображень, дозволяючи сегментувати листя, визначати площу листкової поверхні, оцінювати морфологічні зміни та виявляти ознаки стресу або ураження. Використання глибинного навчання, зокрема моделей детекції YOLOv8 та сегментаційних мереж U-Net або Mask R-CNN, дозволяє створити точну, надійну та масштабовану систему, здатну прогнозувати динаміку росту базиліку та формувати рекомендації щодо оптимізації умов вирощування в режимі реального часу.

3.1 Опис технічного завдання

Метою цього розділу є створення програмної системи для моніторингу стану листя базиліку, вирощеного на гідропонній установці, із застосуванням методів комп'ютерного зору. Система покликана автоматизувати процес виявлення уражень листя на основі цифрових зображень та надавати користувачу зручну візуалізацію стану рослин через веб-інтерфейс. Об'єктом моніторингу є листя базиліку, яке може піддаватися різним захворюванням та стресовим факторам під час гідропонного вирощування.

Система повинна виконувати ряд ключових функцій: завантаження зображень листя, автоматичний аналіз за допомогою моделей машинного навчання, відображення результатів із виділенням уражених ділянок, а також

збереження історії оброблених зображень для подальшого аналізу динаміки росту та стану рослин. Такий підхід забезпечує своєчасне виявлення проблем і дозволяє приймати обґрунтовані рішення щодо догляду за базиліком, підвищуючи ефективність і стабільність процесу гідропонного вирощування.

3.2 Архітектура та технологічна основа системи

Розроблювана система побудована за класичною клієнт-серверною архітектурою з використанням сучасних веб-технологій і передбачає інтеграцію трьох основних компонентів: клієнтської частини, серверної частини та модуля комп’ютерного зору, що відповідає за обробку та аналіз зображень. Така структура забезпечує розподіл функцій між компонентами, дозволяючи досягти високої продуктивності, масштабованості та гнучкості у роботі системи.

Клієнтська частина реалізована у вигляді односторінкового веб-застосунку (SPA) з використанням фреймворку React, що забезпечує швидку реакцію інтерфейсу на дії користувача та комфортну навігацію між розділами. Через REST API клієнтська частина взаємодіє з сервером, дозволяючи користувачам завантажувати фотографії листя базиліку, переглядати результати обробки, отримувати аналітичні дані та здійснювати сортування й фільтрацію інформації. Для візуалізації аналітики та результатів обробки застосовуються технології Canvas, SVG, D3.js або Chart.js, що дозволяє створювати інтерактивні графіки, діаграми та контури уражених ділянок листя. Основні функції клієнтської частини включають завантаження фото у форматах .jpg, .jpeg та .png, перегляд галереї оброблених зображень із пагінацією, відображення результатів аналізу у двох режимах — із виділенням уражених ділянок (BBOX) та інтерактивних контурів, а також виведення статистичних даних, таких як площа ураження листа, кількість уражених зон і загальна аналітика оброблених зображень. Крім того, користувач отримує повідомлення про стан обробки зображень у режимі реального часу за допомогою WebSocket або HTTP polling, що підвищує ефективність взаємодії та прозорість роботи системи.

Серверна частина реалізована на Python 3.x із використанням Django REST Framework, що забезпечує управління запитами, збереження даних, черги завдань та інтеграцію з модулем машинного навчання. Для зберігання даних застосовується база SQLite, а Docker використовується для контейнеризації всіх компонентів системи, що спрощує розгортання та забезпечує портативність. Основні API-інтерфейси включають: POST /api/upload/ для завантаження зображень та повернення результатів у форматі JSON із координатами меж уражень або Base64-зображеннями; GET /api/gallery/ для перегляду списку камер; GET /api/gallery/{cameraId}?from={number}&limit={number} для отримання відеопотоку або галереї конкретної камери; GET /api/gallery/{cameraId}/{photoId} для отримання детальної інформації про окреме зображення; GET /api/stats/ для отримання аналітичних даних, включно з середнім відсотком ураження, частотою захворювань і динамікою росту базиліку.

Модуль комп'ютерного зору відповідає за інференс нейронної мережі, обробку та класифікацію зображень, виявлення зон ураження листя та підготовку структурованих результатів для клієнтської частини. Завдяки такій архітектурі система дозволяє поєднувати високу точність аналізу зручною та інтуїтивною подачею результатів користувачу, забезпечує можливість масштабування на більшу кількість камер та користувачів і створює надійну платформу для подальшого впровадження інтелектуальних алгоритмів прогнозування стану рослин та автоматизованого моніторингу росту базиліку на гідропонній установці.

3.3 Розробка клієнтської частини

Клієнтська частина системи виконує ключову роль у забезпеченні взаємодії користувача з усією інфраструктурою моніторингу росту базиліку. Вона виступає як головний інтерфейс, через який користувач отримує доступ до всіх функцій системи та може спостерігати за станом рослин у режимі реального часу. Клієнтська частина відповідає за відображення даних, що надходять із

серверної частини та модуля комп’ютерного зору, а також за забезпечення інтуїтивного та зручного доступу до функцій, таких як завантаження нових зображень, перегляд галереї оброблених фотографій, візуалізація результатів аналізу та відображення аналітичної інформації у вигляді інтерактивних графіків, діаграм і контурів уражених зон листя.

Для розробки клієнтської частини обрано підхід SPA (Single Page Application) із використанням сучасного фреймворку React. Цей підхід дозволяє забезпечити високу швидкість реагування інтерфейсу на дії користувача, плавну навігацію між різними розділами веб-застосунку, а також інтерактивне відображення результатів аналізу зображень у режимі реального часу. Використання React забезпечує модульність компонентів, що значно спрощує підтримку та масштабування системи, дозволяє швидко додавати нові функції, адаптувати інтерфейс під різні роздільні здатності екранів та розширювати аналітичні можливості системи без порушення роботи вже існуючих модулів.

Клієнтська частина реалізує низку важливих елементів інтерфейсу, серед яких форма для завантаження зображень листя базиліку у форматах .jpg, .jpeg та .png, галерея оброблених фото з підтримкою пагінації, сортування та фільтрації за різними критеріями, блок відображення результатів обробки у вигляді виділених зон ураження та інтерактивних контурів, а також блок аналітики, який дозволяє відображати площу ураження листа, кількість уражених ділянок, середній відсоток ураження та інші статистичні показники по оброблених зображеннях. Окрім цього, реалізовано механізм повідомлень користувача про стан обробки та прогрес виконання завдань, що робить роботу більш прозорою, зручною та ефективною.

На рисунку 3.1 представлена діаграма користувацької взаємодії, яка наочно демонструє основні елементи інтерфейсу та їх взаємозв’язок із серверною частиною та модулем комп’ютерного зору. Така організація дозволяє користувачеві швидко отримувати всю необхідну інформацію, приймати обґрунтовані рішення щодо стану рослин та контролювати процес росту базиліку в умовах гідропонної установки. Клієнтська частина забезпечує не лише

відображення результатів, але й інтерактивну аналітику, що сприяє точнішому прогнозуванню росту рослин, виявленню зон ураження та прийняттю оперативних рішень для підтримки оптимальних умов вирощування.

У підсумку, клієнтська частина виступає важливою складовою інтелектуальної системи моніторингу, поєднуючи функції візуалізації, аналітики та інтерактивного управління, що дозволяє ефективно контролювати процес вирощування базиліку та забезпечувати високий рівень автоматизації в гідропонній системі.

Рисунок 3.1 – Діаграма користувача

Основним елементом інтерфейсу є форма завантаження зображень листя базиліку у форматах JPG, JPEG або PNG (*рис. 3.2)*

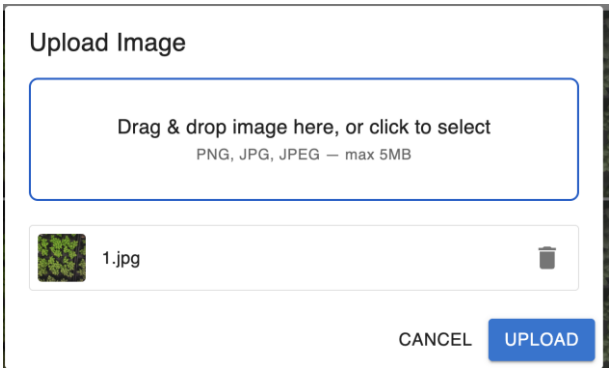


Рисунок 3.2 Форма завантаження зображень

Після надсилання зображення користувач отримує повідомлення про статус обробки. Оброблені зображення зберігаються у галереї (рис 3.3), яка підтримує пагінацію, сортування та фільтрацію за датою або ступенем ураження.

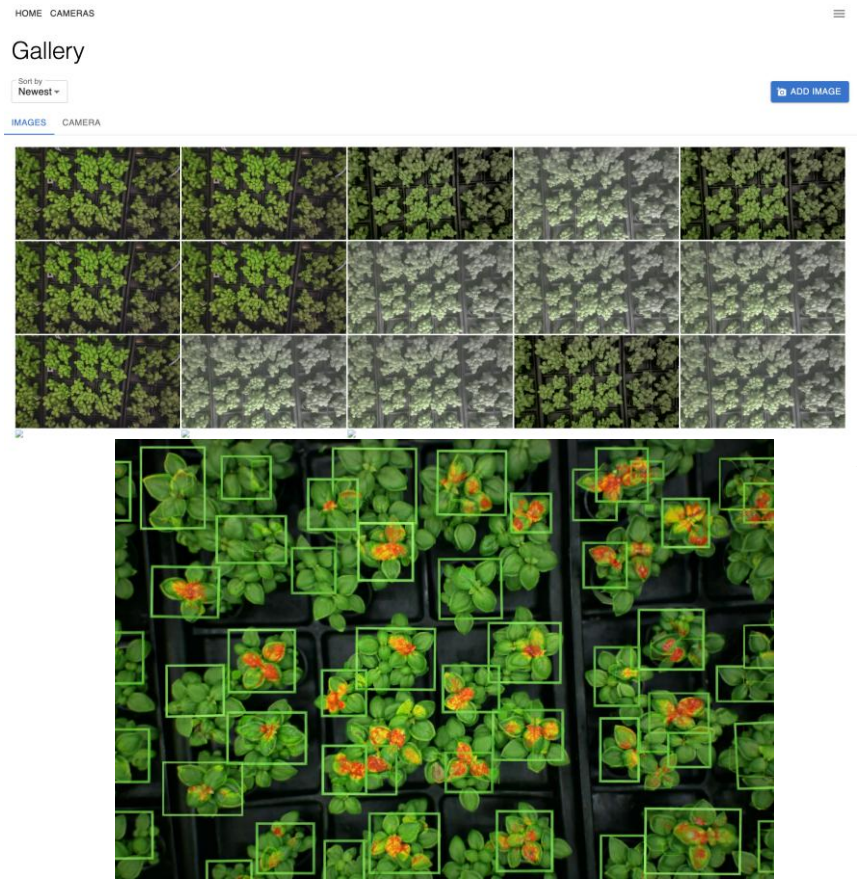


Рисунок 3.3 Галерея

Результати аналізу відображаються у вигляді зображення з виділеними ураженими ділянками за допомогою прямокутних рамок (*малюнок 3.3.3*), а також у вигляді інтерактивних контурів, побудованих із використанням Canvas або SVG. Додатково реалізовано модуль аналітики, який візуалізує такі показники,

як відсоток ураженої площі листя, кількість уражених зон та статистику оброблених зображень.

3.4 Розробка серверної частини та API

Серверна частина веб-платформи для агрегації наукових досліджень у сфері гідропонних ферм виконує роль центрального обчислювального та інтеграційного компонента системи, забезпечуючи координацію взаємодії всіх її модулів. Вона відповідає за обробку запитів від клієнтської частини, управління збереженими даними, інтеграцію з модулем штучного інтелекту для аналізу зображень листя базиліку та формування результатів у зручному для візуалізації форматі. Серверна частина виступає не лише каналом передачі даних, але й центральним компонентом, що забезпечує узгоджене функціонування всієї системи та високу ефективність обробки інформації.

Backend реалізовано з використанням мови програмування Python 3.x та архітектурного підходу REST API, що дозволяє досягти високої масштабованості, модульності та незалежності клієнтської і серверної частин системи. Для обміну даними між компонентами використовується формат JSON, що забезпечує сумісність із різними платформами та зручність інтеграції зовнішніх модулів. Така архітектура дозволяє безперешкодно додавати нові функції, модернізувати алгоритми аналізу та інтегрувати додаткові інтелектуальні сервіси без втручання у роботу клієнтської частини.

Архітектура серверної частини побудована за багаторівневою схемою, що включає кілька логічних шарів, кожен із яких відповідає за певну функціональну складову системи:

- API-шар – прийом HTTP-запитів від клієнтської частини, їх валідація, маршрутизація та передача до відповідних служб.
- Шар бізнес-логіки – реалізація алгоритмів обробки даних, підготовка зображень до аналізу, виклик модуля детекції YOLO, обчислення аналітичних показників, таких як площа ураження листя, кількість уражених ділянок та класифікація типів захворювань.

- Шар доступу до даних – робота з базою даних SQLight, збереження метаданих зображень, результатів аналізу та агрегованої статистики.
- Асинхронний шар обробки – виконання ресурсомістких операцій, таких як аналіз великих масивів зображень, для уникнення блокування основного потоку сервера.

Для реалізації REST API застосовуються сучасні фреймворки, серед яких Django REST Framework, Flask або FastAPI. На практиці перевагу доцільно віддавати FastAPI завдяки високій продуктивності, підтримці асинхронних запитів, автоматичній генерації документації OpenAPI та зручності інтеграції з іншими сервісами. Основні API-ендпоінти включають:

- POST /api/images/upload – завантаження зображень листя базиліку у форматах .jpg, .jpeg або .png; сервер повертає JSON із координатами bounding boxes або Base64-зображення з виділеними ураженими ділянками.
- GET /api/images – отримання списку оброблених зображень із підтримкою пагінації, сортування та фільтрації.
- GET /api/images/{id} – отримання детальної інформації та результатів аналізу конкретного зображення.
- GET /api/statistics – отримання агрегованих даних для формування аналітичних графіків та оцінки динаміки росту, частоти захворювань і площі ураження листя.

Інтеграція з модулем штучного інтелекту реалізована через внутрішній сервісний інтерфейс. Після завантаження зображення сервер передає його до модуля YOLO для детекції уражених ділянок та класифікації типу захворювання, якщо доступна відповідна модель. Результати обробки повертаються у структурованому вигляді, включаючи координати bounding boxes, класи уражень та показники достовірності, що дає змогу серверу виконувати додаткові розрахунки, наприклад визначати площу ураження у відсотках від загальної площі листа.

Для підвищення продуктивності та комфорту користувача система підтримує асинхронну обробку запитів за допомогою зв’язки Redis + Celery, що дозволяє виконувати ресурсоємні обчислення у фоновому режимі та не блокувати роботу основного сервера. Користувач отримує повідомлення про стан обробки через WebSocket або HTTP polling, що забезпечує інтерактивність та мінімізує час очікування без потреби перезавантаження сторінки.

Серверна частина контейнеризується за допомогою Docker, що дозволяє розділити систему на незалежні контейнери для API-сервера, бази даних, Redis та модуля штучного інтелекту. Контейнеризація забезпечує відтворюваність середовища розробки, спрощує розгортання системи у хмарі, полегшує масштабування та підтримку стабільної роботи платформи навіть під високим навантаженням.

У підсумку, серверна частина виступає центральним ядром системи моніторингу, інтегруючи всі компоненти, забезпечуючи надійну обробку даних, взаємодію з модулем штучного інтелекту та формування аналітики, що створює основу для ефективного та автоматизованого контролю росту базиліку у гідропонній установці.

.3.5 Реалізація модуля комп’ютерного зору

Для побудови системи моніторингу росту базиліку на гідропонній установці було використано інтелектуальний модуль комп’ютерного зору DFRobot ESP32-S3 AI Camera (DFR1154). Цей модуль поєднує в собі потужний мікроконтролер ESP32-S3 з вбудованою камерою високої роздільної здатності та апаратним прискоренням обробки зображень, що дозволяє виконувати базові завдання комп’ютерного зору без необхідності підключення до потужного зовнішнього сервера. Завдяки цьому модуль є самодостатнім і забезпечує високу швидкодію при аналізі візуальної інформації без значних затримок.

Програмне забезпечення модуля реалізовано мовою C/C++ із використанням платформи Arduino. Використання Arduino забезпечує легкий доступ до апаратних ресурсів мікроконтролера, дозволяє інтегрувати бібліотеки для

роботи з камерою, сенсорами та засобами комунікації, а також спрощує реалізацію алгоритмів обробки зображень і взаємодії з серверною частиною системи. Arduino-сумісне середовище значно скоротило час розробки та підвищило надійність роботи програмного забезпечення, оскільки воно включає перевірені та стабільні бібліотеки для роботи з периферією, мережевими протоколами та датчиками.

Модуль комп’ютерного зору виконує такі основні функції:

- захоплення RGB-зображень листя базиліку з частотою зйомки, визначеною для тривалого моніторингу;
- попередню обробку зображень, включаючи масштабування, корекцію освітленості та шумозаглушення;
- передачу підготовлених зображень до серверної частини або безпосередньо до інтегрованих алгоритмів штучного інтелекту для детекції зон ураження листя;
- виконання базових алгоритмів комп’ютерного зору на борту пристрою для зменшення обсягу переданих даних та прискорення реакції системи.

Використання апаратного прискорення на ESP32-S3 дозволяє ефективно реалізувати моделі детекції на рівні мікроконтролера, що особливо важливо для автономного моніторингу в умовах обмежених ресурсів та необхідності довготривалого безперервного збору даних. Програмний код модуля та приклади налаштування можна знайти в Додатку А, що дозволяє детально ознайомитися з архітектурою рішення, структурами даних та методами інтеграції з серверною частиною системи.

Завдяки такому поєднанню апаратної платформи ESP32-S3 та програмного забезпечення на Arduino модуль комп’ютерного зору забезпечує ефективний, надійний та масштабований збір і обробку зображень для автоматизованого моніторингу росту та стану базиліку у гідропонних установках.

3.6 Асинхронна обробка та продуктивність системи

Інформаційна система розроблена з урахуванням високих вимог до продуктивності та ефективної обробки даних, що є критично важливим для задач

аналізу зображень у гідропонних установках. Для забезпечення швидкої та безперебійної роботи застосована асинхронна модель взаємодії між клієнтською та серверною частинами, яка дозволяє мінімізувати затримки та підвищити загальну швидкодію системи.

На клієнтській стороні для обміну даними з сервером використовується бібліотека Axios, що забезпечує асинхронне виконання HTTP-запитів до REST API. Завдяки цьому користувач може завантажувати зображення, отримувати результати аналізу та оновлювати інтерфейс у режимі реального часу без блокування взаємодії з веб-застосунком. Крім того, асинхронні запити дозволяють обробляти великі об’єми даних і забезпечують плавність роботи інтерфейсу при одночасному використанні декількох камер.

Серверна частина реалізована на базі фреймворку FastAPI, який підтримує асинхронну обробку запитів за допомогою механізму async/await. Це дозволяє системі одночасно обслуговувати кілька запитів користувачів та камер, включаючи завантаження зображень і отримання результатів аналізу, без затримок у відповіді API. Обчислювально складні операції, пов’язані з роботою моделей комп’ютерного зору та аналізом зображень, винесені в окремий обчислювальний модуль, що дозволяє відокремити процес інференсу від обробки HTTP-запитів. Такий підхід значно зменшує час відповіді системи і підвищує стабільність її роботи при тривалому моніторингу.

Для збереження зображень використовується файлове сховище, а метадані про оброблені зображення та результати аналізу зберігаються у реляційній базі даних SQLite. Така організація даних дозволяє зменшити навантаження на сервер, забезпечити швидкий доступ до інформації та підтримувати ефективну роботу системи у малих і середніх гідропонних установках. Загальна архітектура інтелектуальної системи представлена на малюнку 3.6, де показано взаємодію клієнтської частини, серверної логіки та модуля комп’ютерного зору.

Ця структура забезпечує ефективний і масштабований обмін даними, стабільну обробку зображень та швидке надання аналітичних результатів користувачу, що

є ключовим для оперативного контролю стану рослин та прийняття рішень щодо управління гідропонною установкою.

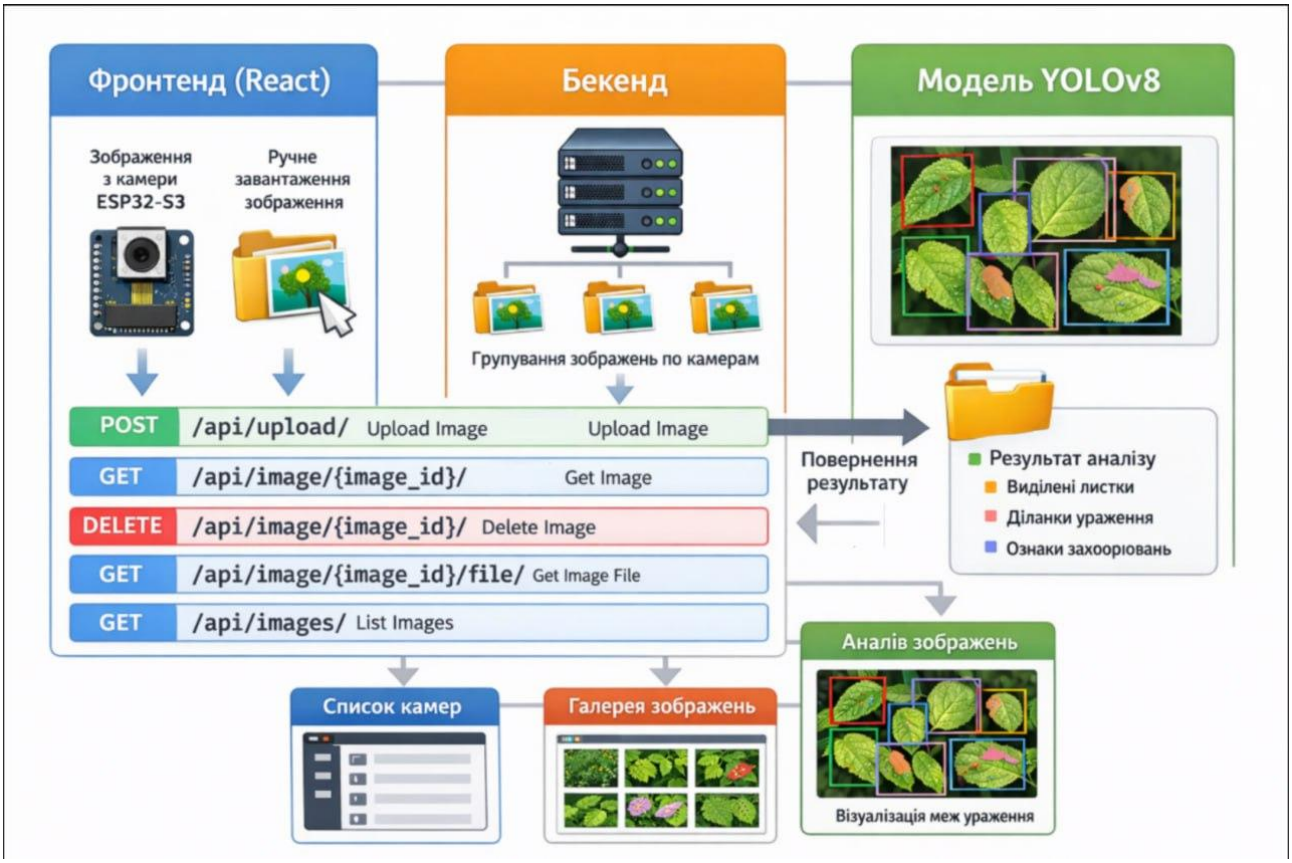


Рисунок 3.4. Загальна схема архітектури інтелектуальної системи

3.7 Контейнеризація та розгортання

Для забезпечення відтворюваності середовища виконання, спрощення процесу розгортання та ізоляції окремих компонентів інформаційної системи застосовано технологію контейнеризації Docker із оркестрацією за допомогою Docker Compose. Такий підхід дозволяє формалізовано описати всі сервіси системи, їх залежності та налаштування, забезпечуючи швидке та надійне розгортання у будь-якому середовищі без необхідності ручної конфігурації.

Система розгортається як набір взаємопов’язаних контейнерів, описаних у файлі docker-compose.yml, що дозволяє централізовано керувати конфігурацією, мережевою взаємодією та життєвим циклом усіх компонентів системи. Це

забезпечує легке масштабування, контроль версій компонентів та мінімізацію ризику конфліктів залежностей між сервісами.

У складі системи виділено два основні сервіси:

- Backend (API) – контейнер із серверною частиною системи, побудований на базі образу python:3.11-slim. У контейнері розгортається FastAPI-застосунок, який надає REST API для роботи з зображеннями, камерами та модулем комп'ютерного зору. Сервер запускається за допомогою Uvicorn, що забезпечує високопродуктивну асинхронну обробку запитів і дозволяє одночасно обслуговувати кілька клієнтських запитів.
- Frontend – контейнер клієнтської частини, реалізований у два етапи. Спочатку виконується збірка React-застосунку у середовищі Node.js, після чого зібрані статичні файли розміщуються на веб-сервері NGINX, що забезпечує швидке та надійне обслуговування клієнтських запитів.

Для збереження даних між перезапусками контейнерів використовується механізм томів (volumes), зокрема для зберігання завантажених зображень та локальної бази даних SQLite. Це гарантує збереження стану системи та історії обробки даних, навіть якщо контейнери зупинено або оновлено. Конфігураційні параметри та змінні середовища винесені у окремий файл .env, що підвищує безпеку, спрощує управління налаштуваннями та дозволяє швидко змінювати конфігурацію без редагування Dockerfile чи коду застосунку.

Розгортання всієї системи виконується однією командою: docker compose up --build -d.

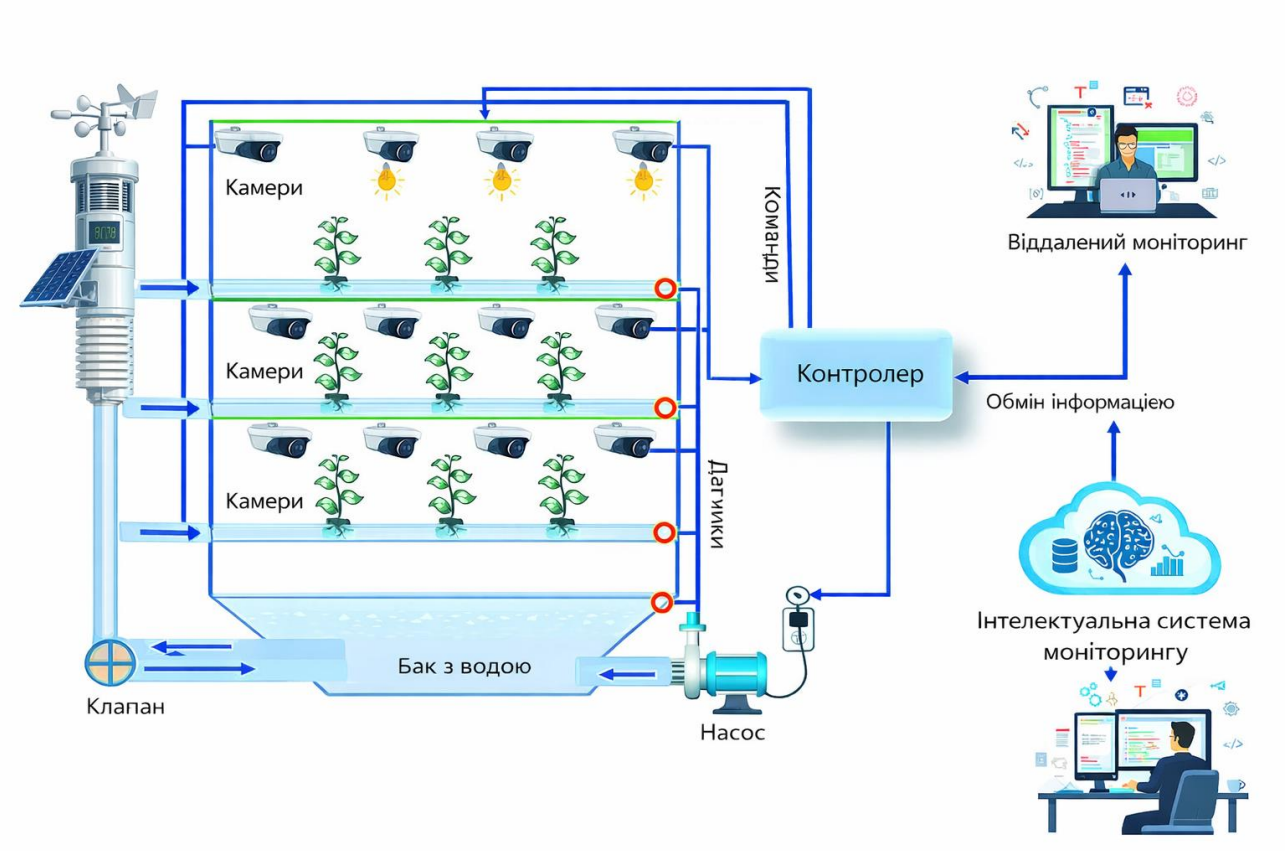
Такий підхід забезпечує автоматизоване створення образів, запуск сервісів у фоновому режимі, просте перенесення системи між різними середовищами та легке масштабування. Крім того, Docker Compose дозволяє безперешкодно додавати нові сервіси, інтегрувати додаткові модулі аналітики або розширювати функціональність системи без порушення стабільності роботи вже існуючих компонентів. Це робить інфраструктуру системи гнучкою, надійною та готовою до подальшого розвитку та інтеграції з іншими сервісами у рамках гідропонних установок.

4. ТЕСТУВАННЯ НА ВИРОБНИЦТВІ

Дослідження, описані в даному розділі, проводилися в умовах реального виробничого середовища — агропромислового підприємства, що спеціалізується на вирощуванні рослин у тепличних умовах із застосуванням гідропонних технологій. Основною культурою, на якій проводилось тестування системи, є базилік.

4.1 Структура підприємства

Підприємство складається з комплексу гідропонних стендів закритого типу. Кожна теплиця є автономною виробничою зоною, але водночас інтегрована в загальну систему моніторингу та управління. Усередині теплиць розміщені багаторівневі стелажі, призначені для компактного та ефективного розміщення гідропонних установок.



Рисинук 4.1 Комплекс гідропонних стенді

Стелажі складаються з окремих поличок (рівнів), на яких розташовуються лотки з рослинами. Така ієрархічна структура (теплиця → стелаж → поличка)

дозволяє чітко зонувати простір, а також прив’язувати дані моніторингу до конкретної фізичної ділянки виробництва (малюнок 4.1)

4.2 Система збору даних та метеостанція

Для реалізації серверної частини системи застосовується сучасний стек технологій, що забезпечує високу продуктивність, масштабованість і надійність обробки даних. Основою бекенду є фреймворк FastAPI, який дозволяє створювати високопродуктивні REST API з асинхронною обробкою запитів на базі механізму async/await. Сервер працює на ASGI-сервері Uvicorn, що забезпечує одночасне обслуговування великої кількості запитів від клієнтів та підтримує обробку даних у режимі реального часу, що особливо важливо для моніторингу стану рослин у гідропонних установках.

Для роботи з базою даних використовується SQLite у поєднанні з SQLAlchemy — потужним ORM, що дозволяє об’єктно-орієнтовано взаємодіяти з реляційними даними, забезпечує легке формування запитів та підвищує гнучкість у роботі з даними. Міграції бази даних виконуються за допомогою Alembic, що дозволяє контролювати зміни структури таблиць та підтримувати сумісність даних між різними версіями системи. Для валідації та серіалізації вхідних і вихідних даних використовується Pydantic, що гарантує коректність форматів і запобігає помилкам під час обробки запитів.

Клієнтська частина системи реалізована за сучасним підходом SPA (Single Page Application) з використанням React 19 та TypeScript, що забезпечує модульність, підтримку типізації та швидку реакцію інтерфейсу на дії користувача. Для стилізації застосовується MUI 7, а для управління станом додатку та асинхронними запитами використовуються бібліотеки Axios і React Query. CSS-modules забезпечують локальну ізоляцію стилів, що спрощує підтримку великих проєктів і унеможливлює конфлікти стилів. Для віддачі зібраних статичних сторінок використовується веб-сервер NGINX, що гарантує стабільну та швидку роботу фронтенду, а також зручне обслуговування великих обсягів даних.

На підприємстві встановлена локальна метеостанція, яка здійснює безперервний моніторинг ключових параметрів мікроклімату, таких як:

- температура повітря;
- відносна вологість повітря;
- температура поживного розчину;
- рівень освітленості;
- концентрація CO_2 (як додатковий параметр, що може впливати на швидкість росту рослин).

Всі дані з метеостанції в режимі реального часу передаються на центральний комп’ютер підприємства, який виконує роль серверного вузла. На цьому вузлі розгорнуто програмний застосунок, що забезпечує агрегацію, зберігання та обробку даних, інтегруючи інформацію від метеостанції та модулю комп’ютерного зору. Це дозволяє формувати детальні аналітичні звіти, візуалізувати стан рослин, аналізувати динаміку росту та своєчасно приймати рішення щодо оптимізації умов вирощування базиліку на гідропонній установці.

Поєднання сучасного фронтенду, надійного бекенду, інтеграції з метеоданими та модулем комп’ютерного зору забезпечує повний цикл моніторингу — від збору даних до аналізу та візуалізації результатів. Така архітектура робить систему ефективним інструментом для автоматизованого управління мікрокліматом, контролю росту та стану рослин, а також підвищує точність прийняття рішень і оптимізує продуктивність гідропонної ферми. Система легко масштабується та може бути розширена для роботи з додатковими камерами, датчиками та аналітичними модулями, що забезпечує її довгострокову ефективність та універсальність у використанні.

Для реалізації серверної частини системи застосовується сучасний стек технологій, що забезпечує високу продуктивність, масштабованість і надійність обробки даних. Основою бекенду є фреймворк FastAPI, який дозволяє створювати високопродуктивні REST API з асинхронною обробкою запитів на

базі механізму async/await. Сервер працює на ASGI-сервері Uvicorn, що забезпечує одночасне обслуговування великої кількості запитів від клієнтів та підтримує обробку даних у режимі реального часу, що особливо важливо для моніторингу стану рослин у гідропонних установках.

4.3 Відеомоніторинг та комп'ютерний зір

Ключовим компонентом системи є мережа відеокамер високої роздільної здатності, встановлених безпосередньо над кожним рівнем стелажів із рослинами. Така конфігурація забезпечує багатошаровий візуальний контроль, дозволяючи детально відстежувати стан листя, розвиток рослин та загальний стан культури на всіх рівнях установки.

Користувач отримує розширені можливості управління та моніторингу, зокрема:

- логічно маркувати кожну камеру відповідно до її фізичного розташування (теплиця, стелаж, рівень), що спрощує орієнтування у великій системі;
- переглядати централізовану галерею зображень з усіх камер у реальному часі або за історичними даними;
- виконувати вибірковий аналіз зображень з конкретної камери за визначений період, що дозволяє відстежувати динаміку росту та зміну стану рослин.

Отримані відео- та фотоматеріали передаються безпосередньо до центрального застосунку, де вони інтегруються з модулями штучного інтелекту для автоматизованого аналізу. Алгоритми комп'ютерного зору здійснюють сегментацію листкової поверхні, оцінюють площу здорових і уражених ділянок, а також визначають потенційні проблеми, включаючи:

- дефіцит або дисбаланс поживних речовин;
- невідповідний температурний режим;
- надмірну або недостатню вологість повітря та субстрату;
- інші стресові фактори, які можуть впливати на інтенсивність росту.

У разі виявлення відхилень система автоматично генерує сповіщення для користувача, включаючи детальну інформацію про тип проблеми, рівень

критичності та рекомендації щодо корекції. Це забезпечує швидке реагування, дозволяє запобігти втратам і підвищує ефективність управління мікрокліматом і умовами вирощування базиліку (рис 4.2).

Такий технічний підхід дозволяє поєднувати реальний час моніторингу з історичною аналітикою, забезпечує масштабованість системи та інтеграцію з іншими компонентами інтелектуальної ферми, включно з датчиками мікроклімату та модулем прогнозування росту рослин.

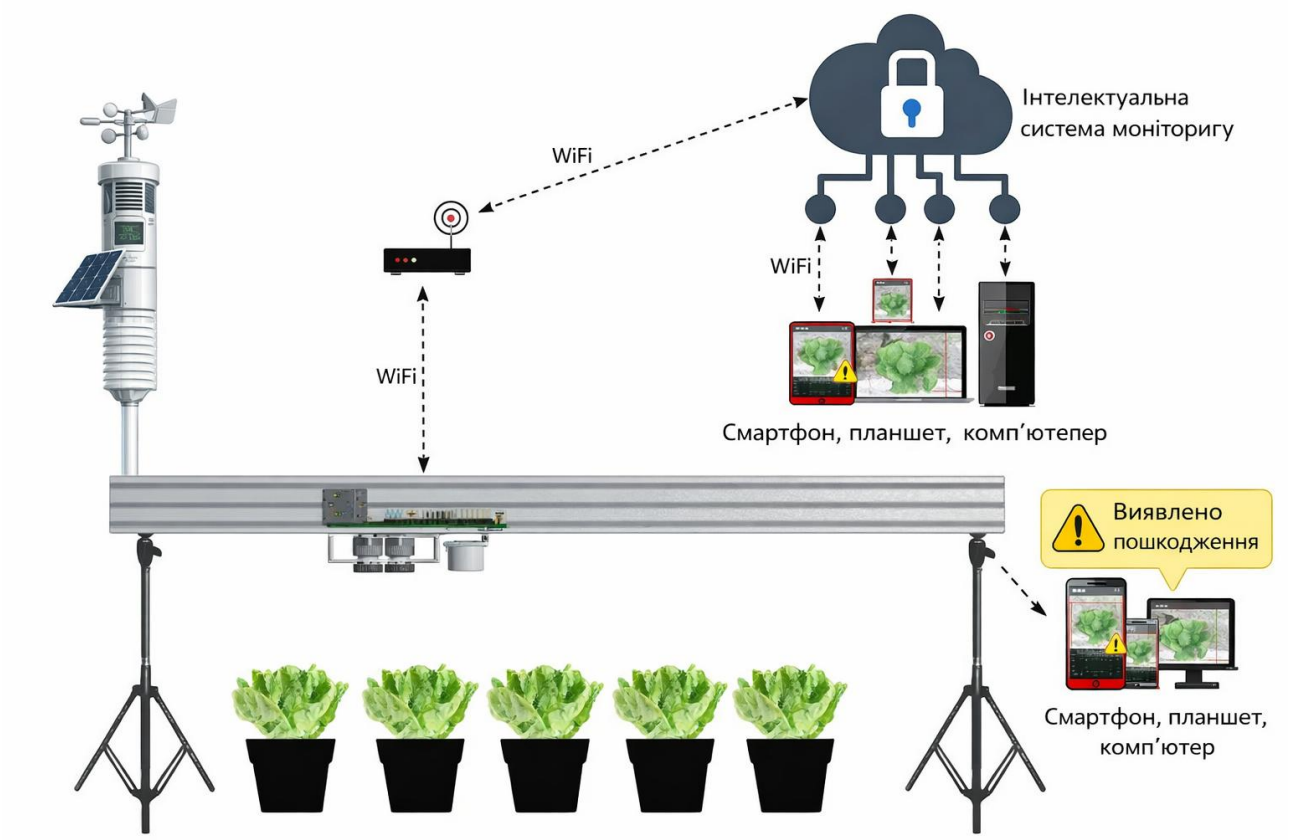


Рисунок 4.2 Узагальнена структурна схема тепличного підприємства та взаємодії камер, застосунку і моделей штучного інтелекту

4.2 Тестування системи у виробничих умовах

Тестування системи моніторингу росту базиліку на гідропонній установці проводиться безпосередньо у виробничих умовах для оцінки її працездатності, точності та надійності. Основною метою цього етапу є перевірка інтеграції всіх

компонентів системи — камер, модулів комп’ютерного зору, серверної та клієнтської частин, а також взаємодії з локальною метеостанцією.

Процес тестування включає такі ключові етапи:

- перевірка коректності передачі зображень з усіх камер до центрального серверного вузла;
- оцінка точності алгоритмів детекції та сегментації листя та уражених ділянок;
- контроль продуктивності системи при одночасній обробці даних з кількох камер;
- тестування інтерфейсу користувача, включаючи завантаження зображень, перегляд галереї, відображення аналітики та сповіщень про стан рослин;
- перевірка коректності інтеграції з метеостанцією та відображення реальних показників мікроклімату у системі.

У ході тестування оцінюються такі параметри системи:

- швидкість обробки зображень та оновлення даних у інтерфейсі;
- точність визначення уражених ділянок та правильність класифікації типів проблем;
- стабільність роботи при безперервному зборі та обробці даних упродовж кількох днів;
- надійність сповіщень та повідомлень для користувача;
- коректність збереження та відображення історичних даних у галереї та статистичних звітах.

Результати тестування дозволяють виявити можливі вузькі місця в роботі системи, оптимізувати параметри алгоритмів та налаштування серверного і клієнтського програмного забезпечення. На основі цих даних здійснюється вдосконалення системи для забезпечення стабільного та точного моніторингу рослин у реальних виробничих умовах, що є критично важливим для ефективного управління гідропонною установкою.

4.2.1 Характеристика використаної камери

Для відеозахоплення у системі моніторингу під час тестування застосовувалася камера на базі мікроконтролера ESP32-S3 AI Camera. Основною перевагою такого апаратного рішення є висока гнучкість програмування, компактність та легкість інтеграції з існуючими модулями керування та системою комп'ютерного зору. Камера підтримує безпосереднє підключення до мережі, передачу зображень у реальному часі та апаратне прискорення базових обчислень, що дозволяє виконувати попередню обробку зображень без залучення потужного серверного обладнання.

Водночас у процесі експлуатації було виявлено деякі технічні обмеження. Зокрема, камера має відносно обмежену роздільну здатність, що знижує деталізацію зображень та ускладнює точне розпізнавання дрібних ознак ураження листя, особливо на ранніх стадіях розвитку рослин. Крім того, обмежена світлочутливість сенсора може впливати на якість зображень за умов недостатнього освітлення, а вузький динамічний діапазон іноді призводить до втрати деталей у зонах з різким контрастом.

Незважаючи на ці обмеження, камера забезпечує достатню якість зображень для первинного аналізу та моніторингу стану рослин, особливо в поєднанні з алгоритмами сегментації та детекції на базі нейронних мереж. Додатково, її компактні розміри та низьке енергоспоживання роблять її оптимальним варіантом для масштабування системи та встановлення на кожній поличці гідропонної установки.

Як технічне вдосконалення системи можна розглянути використання камер з підвищеною роздільною здатністю або додатковою інфрачервоною підсвіткою, що дозволить підвищити точність детекції та раннє виявлення ознак стресу або хвороб у рослин.

4.2.2 Вплив освітлення та адаптація параметрів

Якість відеозахоплення та точність аналізу зображень у тепличних умовах значною мірою залежить від стабільності освітлення. У реальних виробничих

умовах часто спостерігаються коливання інтенсивності штучного світла або короткочасні перебої з електропостачанням, що негативно впливає на контрастність і деталізацію зображень та ускладнює виявлення дрібних ознак стану рослин.

Використана камера має можливість програмного регулювання критичних параметрів захоплення зображення, зокрема експозиції, чутливості сенсора та балансу білого. Для забезпечення стабільної якості зображень у широкому діапазоні умов освітлення було реалізовано алгоритм адаптивної корекції параметрів камери. Система аналізує вхідне зображення, визначає рівень освітленості сцени та автоматично підлаштовує експозицію і чутливість сенсора, підтримуючи чіткість і контраст навіть при низькому або нестабільному освітленні.

Технічні переваги цього підходу:

- стабільне отримання зображень високої якості за будь-якого рівня освітленості;
- автоматичне фіксування періодів недостатнього освітлення, що дозволяє користувачу аналізувати роботу системи освітлення;
- можливість оптимізації режимів штучного освітлення для економії енергії та підтримки оптимальних умов росту рослин.

Завдяки впровадженню адаптивної корекції параметрів камери система забезпечує надійний та точний моніторинг стану рослин, підвищує ефективність комп’ютерного зору та дозволяє приймати обґрунтовані рішення щодо управління мікрокліматом у теплиці.

4.3 Висновки до розділу

У результаті проведеного тестування на виробництві було підтверджено практичну придатність розробленої системи. Незважаючи на апаратні обмеження використаної камери, програмні методи адаптації дозволили досягти стабільної якості вхідних даних.

У цілому було створено актуальну систему моніторингу росту базиліку на гідропонній установці з використанням методів комп’ютерного зору. Система забезпечує комплексний контроль мікроклімату та візуального стану рослин і може бути використана як основа для подальшого розвитку інтелектуальних аграрних рішень.

Висновки

1. У результаті проведеного дослідження було розроблено комплексну інтелектуальну систему моніторингу росту базиліку на гідропонній установці, яка поєднує апаратні засоби відеоспостереження, модулі комп'ютерного зору та веб-платформу для збору, обробки та візуалізації даних.
2. Для реалізації моніторингу стану листя застосовано модулі комп'ютерного зору на базі камер ESP32-S3 із апаратним прискоренням обробки зображень. Було розроблено алгоритми детекції та сегментації, які дозволяють визначати площу листкової поверхні, ідентифікувати уражені ділянки та оцінювати загальний стан рослин у режимі реального часу.
3. Впроваджено асинхронну архітектуру клієнт-серверної взаємодії, що забезпечує швидку обробку запитів та високу продуктивність системи. Використання FastAPI та Uvicorn для бекенду, React та TypeScript із MUI і React Query для фронтенду, а також SQLite із SQLAlchemy для зберігання даних дозволяє досягти масштабованості, модульності та надійності системи.
4. Для забезпечення гнучкої адаптації до умов освітлення було реалізовано алгоритм автоматичного регулювання параметрів камери (експозиції, чутливості та балансу білого) на основі аналізу вхідного зображення. Це дозволяє підтримувати високу якість зображень навіть за нестабільного або недостатнього освітлення.
5. Розроблено систему інтеграції з локальною метеостанцією, яка здійснює моніторинг температури повітря, вологості, температури поживного розчину, рівня освітленості та концентрації CO_2. Зібрані дані об'єднуються з результатами комп'ютерного зору, що дозволяє формувати аналітичні звіти, відстежувати динаміку росту рослин та оптимізувати умови гідропонного вирощування.
6. Для забезпечення відтворюваності середовища виконання та зручності розгортання система контейнеризована за допомогою Docker із Docker

Compose. Це дозволяє легко масштабувати рішення, додавати нові сервіси та забезпечує стабільну роботу в різних середовищах.

7. У ході тестування системи у виробничих умовах підтверджено її ефективність: алгоритми детекції та сегментації дозволяють ідентифікувати уражені ділянки листя з високою точністю, система коректно реагує на зміни освітлення та передає дані користувачу у реальному часі, що забезпечує оперативний контроль за станом рослин.
8. Запропонована інтелектуальна система є універсальною платформою для автоматизованого моніторингу гідропонних культур, що поєднує збір візуальної та метеодані, аналітику стану рослин та інтерактивне відображення результатів у зручному веб-інтерфейсі.
9. Отримані результати створюють основу для подальшого розвитку системи, включаючи: інтеграцію додаткових сенсорів, підвищення роздільної здатності камер, впровадження алгоритмів прогнозування росту та захворювань рослин із використанням методів штучного інтелекту.

Список використаних джерел

Список використаних джерел

1. Salka T. D., Hanafi M. B., Rahman S. M. S. A. A., et al. Plant leaf disease detection and classification using convolution neural networks model: a review. *Springer*, 2025. DOI: 10.1007/s10462-025-11234-6. URL: https://link.springer.com/article/10.1007/s10462-025-11234-6

2. Pascal I. A systematic review of deep learning techniques for plant disease detection. *Springer*, 2024. DOI: 10.1007/s10462-024-10944-7. URL: https://link.springer.com/article/10.1007/s10462-024-10944-7

3. Nyawose T. A review on the detection of plant disease using machine learning and image processing techniques. *MDPI Vision*, 2025. DOI: 10.3390/vision11100326. URL: https://www.mdpi.com/2313-433X/11/10/326

4. Nguyen D. T. Improving YOLO-Based Plant Disease Detection Using Advanced Deep Learning Techniques. *MDPI Agronomy*, 2025. DOI: 10.3390/agronomy7090271. URL: https://www.mdpi.com/2624-7402/7/9/271

5. Scientific Reports (Nature). Machine learning insights for sustainable hydroponic cultivation and growth monitoring of Allium cepa using smart hydro kit. 2024. DOI: 10.1038/s41598-024-82920-8. URL: https://www.nature.com/articles/s41598-024-82920-8

6. Central Asian Journal of Mathematical Theory and Computer Sciences. Smart Hydroponic Growth Optimisation System with Real–Time Monitoring and Control with CNN. DOI: уточнюється. URL: https://cajmtcs.casjournal.org/index.php/CAJMTCS/article/view/823

7. Information Technology and Society. Plant Monitoring System: ІІІІ-система для розумного моніторингу рослин. 2025. DOI: 10.32689/maup.it.2025.1.7. URL: https://journals.maup.com.ua/index.php/it/article/view/4805

8. Development of IoT Smart Greenhouse System for Hydroponic Gardens. *arXiv Preprint*, 2023. URL: https://arxiv.org/abs/2305.01189

9. Multimodal Data Integration for Sustainable Indoor Gardening: Tracking Anyplant. *arXiv Preprint*, 2025. URL: https://arxiv.org/abs/2503.21932

10. International Journal for Research in Applied Science and Engineering Technology. Smart Hydroponic System and Monitoring of Plant's Health through Machine Learning, 2023. DOI: 10.22214/ijraset.2023.54443. URL: https://www.ijraset.com/research-paper/smart-hydroponic-system-and-monitoring-of-plants-health

11. International Journal of Innovative Science and Research Technology. Smart Hydroponic Greenhouse Using Business Intelligence Tools and Deep Learning, 2025. DOI: 10.38124/ijisrt/25feb1515. URL: https://www.ijisrt.com/assets/upload/files/IJISRT25FEB1515.pdf

12. International Journal of Intelligent Systems and Applications in Engineering. Machine Learning Based Approach for Crop Growth Monitoring in Hydroponics, 2023. URL: https://ijisae.org/index.php/IJISAE/article/view/3944

13. Schneider F., Swiatek J., Jelali M. Detection of Growth Stages of Chilli Plants in a Hydroponic Grower Using Machine Vision and YOLOv8 Deep Learning Algorithms. *MDPI Sustainability*, 2024. DOI: 10.3390/su16156420. URL: https://www.mdpi.com/2071-1050/16/15/6420

14. Yin-Syuen Tong, Tou-Hong Lee, Kin-Sam Yen. Deep Learning for Image-Based Plant Growth Monitoring: A Review. *IJETI*, 2022. DOI: 10.46604/ijeti.2022.8865. URL: https://ojs.imeti.org/index.php/IJETI/article/view/8865

15. Rathor A. S., Choudhury S., Sharma A., Nautiyal P., Shah G. Empowering vertical farming through IoT and AI-Driven technologies: A comprehensive review. *Heliyon*, 2024. DOI: 10.1016/j.heliyon.2024.e34998. URL: https://linkinghub.elsevier.com/retrieve/pii/S2405844024110298

16. Zheng X., Shao Z., Chen Y., Zeng H., Chen J. MSPB-YOLO: High-Precision Detection Algorithm of Multi-Site Pepper Blight Disease Based on Improved YOLOv8. *MDPI Agronomy*, 2025. DOI: 10.3390/agronomy15040839. URL: https://www.mdpi.com/2073-4395/15/4/839

17. PeerJ Computer Science. IoT-based control and monitoring system for hydroponic plant growth, 2025. DOI: 10.7717/cs.2763. URL: https://peerj.com/articles/cs-2763/

18. Koniukhov V. D., Morgun O. M., Nemchenko K. E. Machine Learning and Image Segmentation in Computer Vision. 2024. DOI: 10.15588/1607-3274-2024-4-10. URL: https://ric.zp.edu.ua/article/view/316201

19. Nazarkevych M. A., Oleksiv N. T. Система розпізнавання об’єктів на основі YOLO. 2024. DOI: 10.23939/ujit2024.01.120. URL: https://science.lpnu.ua/ujit/all-volumes-and-issues/volume-6-number-1/object-recognition-system-based-yolo-model-and

20. Civadyts P., Skrypnyk T., Voznyuk L. Порівняння методів виявлення об’єктів у CV. 2024. DOI: 10.31891/2307-5732-2024-333-2-41. URL: https://heraldts.khmnu.edu.ua/index.php/heraldts/article/view/145

Додаток А

main.ino:

```
#include "esp_camera.h"

#include <WiFi.h>
#include <Preferences.h>
#include "FS.h"
#include "SD.h"
#include "driver/rtc_io.h"
#include <WiFiClientSecure.h>
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
#include "Base64.h"
#define SD_CARD_CS   10

#define PWDN_GPIO_NUM     -1
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM     5
#define Y9_GPIO_NUM       4
#define Y8_GPIO_NUM       6
#define Y7_GPIO_NUM       7
#define Y6_GPIO_NUM       14
#define Y5_GPIO_NUM       17
#define Y4_GPIO_NUM       21
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM       16
#define VSYNC_GPIO_NUM    1
#define HREF_GPIO_NUM     2
#define PCLK_GPIO_NUM     15
#define SIOD_GPIO_NUM  8
#define SIOC_GPIO_NUM  9

#define LED_GPIO_NUM  47

#define TIME_TO_SLEEP  60 // Shooting interval time (S)
#define uS_TO_S_FACTOR 1000000ULL // a multiplier for
converting to microseconds
#define SLEEP_DURATION (TIME_TO_SLEEP * uS_TO_S_FACTOR)

String myDeploymentID = "YOUR_API_KEY";
String myMainFolderName = "photos";

//Variables for Timer/Millis.
```

```
unsigned long previousMillis = 0;
const int Interval = 20000; //--> Capture and Send a photo
every 20 seconds.

WiFiClientSecure client;

// This subroutine is to test the connection to
"script.google.com".
void Test_Con() {
  const char* host = "script.google.com";
  while(1) {
    Serial.println("-----------");
    Serial.println("Connection Test...");
    Serial.println("Connect to " + String(host));

    client.setInsecure();

    if (client.connect(host, 443)) {
      Serial.println("Connection successful.");
      Serial.println("-----------");
      client.stop();
      break;
    } else {
      Serial.println("Connected to " + String(host) + "
failed.");
      Serial.println("Wait a moment for reconnecting.");
      Serial.println("-----------");
      client.stop();
    }

    delay(1000);
  }
}

RTC_DATA_ATTR int photo_count = 0; // Use RTC memory to save
counts (lost in case of power failure)

Preferences preferences;
const char* ssid;
const char* password;


void startCameraServer();
void setupLedFlash(int pin);
```

```
void uploadPhotoToDrive(uint8_t *image, size_t len) {
  HTTPClient http;
  http.begin("https://script.google.com/macros/s/YOUR_API_KEY/
exec?folder=photos");
  http.addHeader("Content-Type", "image/jpeg");

  int httpResponseCode = http.POST(image, len);
  if (httpResponseCode > 0) {
    Serial.printf("Uploaded to Google Drive. Response: %d\n",
httpResponseCode);
  } else {
    Serial.printf("Upload failed: %s\n",
http.errorToString(httpResponseCode).c_str());
  }
  http.end();
}

bool initSDCard() {
  if (!SD.begin(SD_CARD_CS)) {
    return false;
  }
  uint8_t cardType = SD.cardType();
  return cardType != CARD_NONE;
}

void takePhotoAndSave() {
  camera_fb_t * fb = esp_camera_fb_get();
  if (!fb) {
    Serial.println("Failed to obtain the image.");
    return;
  }

  String path = "/photo_" + String(photo_count) + ".jpg";
  fs::FS &fs = SD;
  File file = fs.open(path.c_str(), FILE_WRITE);
  if (!file) {
    Serial.println("Save failed");
  } else {
    file.write(fb->buf, fb->len);
    Serial.println("Photo saving path: " + path);
  }
  file.close();
  uploadPhotoToDrive(fb->buf, fb->len);
```

```
  esp_camera_fb_return(fb);

  photo_count++; // The number of the next picture
}

void connectToWiFi(const char* ssid, const char* password) {
  WiFi.begin(ssid, password);
  Serial.printf("Connecting to WiFi: %s\n", ssid);
  int retries = 0;
  while (WiFi.status() != WL_CONNECTED && retries < 20) {
    delay(500);
    Serial.print(".");
    retries++;
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\nWiFi connected!");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
  } else {
    Serial.println("\nFailed to connect to WiFi.");
  }
}

void initWiFi() {
  WiFi.disconnect(true, true);  // повний reset WiFi
  delay(200);
  WiFi.mode(WIFI_STA);
  delay(200);

  preferences.begin("wifi", false);
  String savedSSID = preferences.getString("ssid", "");
  String savedPASS = preferences.getString("password", "");

  if (savedSSID.length() > 0 && savedPASS.length() > 0) {
    Serial.println("Found saved WiFi credentials.");
    connectToWiFi(savedSSID.c_str(), savedPASS.c_str());

    if (WiFi.status() == WL_CONNECTED) {
      return;
    } else {
      Serial.println("Stored credentials failed. Please enter
new credentials.");
    }
```

```
  } else {
    Serial.println("No WiFi credentials found. Please
enter:");
  }

  while (Serial.available()) Serial.read();

  Serial.println("Enter SSID: ");
  while (Serial.available() == 0) delay(10);
  String inputSSID = Serial.readStringUntil('\n');
  inputSSID.trim();

  Serial.println("Enter Password: ");
  while (Serial.available() == 0) delay(10);
  String inputPASS = Serial.readStringUntil('\n');
  inputPASS.trim();

  connectToWiFi(inputSSID.c_str(), inputPASS.c_str());



  if (WiFi.status() == WL_CONNECTED) {
    preferences.putString("ssid", inputSSID.c_str());
    preferences.putString("password", inputPASS.c_str());
    Serial.println("WiFi credentials saved.");
  } else {
    Serial.println("Failed to connect. Credentials not
saved.");
  }

  preferences.end();
}

void initCamera(){
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
```

```
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.frame_size = FRAMESIZE_UXGA;
  config.pixel_format = PIXFORMAT_JPEG;  // for streaming
  //config.pixel_format = PIXFORMAT_RGB565; // for face
detection/recognition
  config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
  config.fb_location = CAMERA_FB_IN_PSRAM;
  config.jpeg_quality = 12;
  config.fb_count = 1;

  // if PSRAM IC present, init with UXGA resolution and higher
JPEG quality
  //                     for larger pre-allocated frame
buffer.
  if (config.pixel_format == PIXFORMAT_JPEG) {
    if (psramFound()) {
      config.jpeg_quality = 10;
      config.fb_count = 2;
      config.grab_mode = CAMERA_GRAB_LATEST;
    } else {
      // Limit the frame size when PSRAM is not available
      config.frame_size = FRAMESIZE_SVGA;
      config.fb_location = CAMERA_FB_IN_DRAM;
    }
  } else {
    // Best option for face detection/recognition
    config.frame_size = FRAMESIZE_240X240;
  }

  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }
```

```
  sensor_t *s = esp_camera_sensor_get();
  // initial sensors are flipped vertically and colors are a
bit saturated
  if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1);        // flip it back
    s->set_brightness(s, 1);   // up the brightness just a bit
    s->set_saturation(s, -2);  // lower the saturation
  }
  // drop down frame size for higher initial frame rate
  if (config.pixel_format == PIXFORMAT_JPEG) {
    s->set_framesize(s, FRAMESIZE_QVGA);
  }
}



void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  delay(4000);
  Serial.println();

  initCamera();
  setupLedFlash(LED_GPIO_NUM);
  delay(500);
  initWiFi();
  startCameraServer();

  if (!initSDCard()) {
    Serial.println("The initialization of the SD card
failed.");
    return;
  }
  pinMode(3,OUTPUT);
  digitalWrite(3,HIGH);
  takePhotoAndSave();
  //delay(500);
  digitalWrite(3,LOW);
  Serial.println("Get ready to enter deep sleep.");
  // esp_sleep_enable_timer_wakeup(SLEEP_DURATION);
  // esp_deep_sleep_start();
}
```

```
void loop() {
  unsigned long now = millis();

  if (now - lastPhotoTime >= SLEEP_DURATION ) {
    lastPhotoTime = now;

    pinMode(3, OUTPUT);
    digitalWrite(3, HIGH);
    takePhotoAndSave();
    digitalWrite(3, LOW);

    Serial.println("Photo saved (timer).");
  }

  delay(50);
}
```

App_httpd.cpp:

```
// Copyright 2015-2016 Espressif Systems (Shanghai) PTE LTD
#include "esp_http_server.h"
#include "esp_timer.h"
#include "esp_camera.h"
#include "img_converters.h"
#include "fb_gfx.h"
#include "esp32-hal-ledc.h"
#include "sdkconfig.h"
#include "camera_index.h"

#if defined(ARDUINO_ARCH_ESP32) &&
defined(CONFIG_ARDUHAL_ESP_LOG)
#include "esp32-hal-log.h"
#endif

// Enable LED FLASH setting
#define CONFIG_LED_ILLUMINATOR_ENABLED 1

// LED FLASH setup
#if CONFIG_LED_ILLUMINATOR_ENABLED

#define LED_LEDC_GPIO            47  //configure LED pin
#define CONFIG_LED_MAX_INTENSITY 255

int led_duty = 0;
```

```
bool isStreaming = false;

#endif

typedef struct {
  httpd_req_t *req;
  size_t len;
} jpg_chunking_t;

#define PART_BOUNDARY "123456789000000000000987654321"
static const char *_STREAM_CONTENT_TYPE = "multipart/x-mixed-
replace;boundary=" PART_BOUNDARY;
static const char *_STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY
"\r\n";
static const char *_STREAM_PART = "Content-Type:
image/jpeg\r\nContent-Length: %u\r\nX-Timestamp:
%d.%06d\r\n\r\n";

httpd_handle_t stream_httpd = NULL;
httpd_handle_t camera_httpd = NULL;

typedef struct {
  size_t size;   //number of values used for filtering
  size_t index;  //current value index
  size_t count;  //value count
  int sum;
  int *values;  //array to be filled with values
} ra_filter_t;

static ra_filter_t ra_filter;

static ra_filter_t *ra_filter_init(ra_filter_t *filter, size_t
sample_size) {
  memset(filter, 0, sizeof(ra_filter_t));

  filter->values = (int *)malloc(sample_size * sizeof(int));
  if (!filter->values) {
    return NULL;
  }
  memset(filter->values, 0, sample_size * sizeof(int));

  filter->size = sample_size;
  return filter;
}
```

```
#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
static int ra_filter_run(ra_filter_t *filter, int value) {
  if (!filter->values) {
    return value;
  }
  filter->sum -= filter->values[filter->index];
  filter->values[filter->index] = value;
  filter->sum += filter->values[filter->index];
  filter->index++;
  filter->index = filter->index % filter->size;
  if (filter->count < filter->size) {
    filter->count++;
  }
  return filter->sum / filter->count;
}
#endif

#if CONFIG_LED_ILLUMINATOR_ENABLED
void enable_led(bool en) {  // Turn LED On or Off
  int duty = en ? led_duty : 0;
  if (en && isStreaming && (led_duty >
CONFIG_LED_MAX_INTENSITY)) {
    duty = CONFIG_LED_MAX_INTENSITY;
  }
  ledcWrite(LED_LEDC_GPIO, duty);
  //ledc_set_duty(CONFIG_LED_LEDC_SPEED_MODE,
CONFIG_LED_LEDC_CHANNEL, duty);
  //ledc_update_duty(CONFIG_LED_LEDC_SPEED_MODE,
CONFIG_LED_LEDC_CHANNEL);
  log_i("Set LED intensity to %d", duty);
}
#endif

static esp_err_t bmp_handler(httpd_req_t *req) {
  camera_fb_t *fb = NULL;
  esp_err_t res = ESP_OK;
#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
  uint64_t fr_start = esp_timer_get_time();
#endif
  fb = esp_camera_fb_get();
  if (!fb) {
    log_e("Camera capture failed");
    httpd_resp_send_500(req);
```

```
    return ESP_FAIL;
  }

  httpd_resp_set_type(req, "image/x-windows-bmp");
  httpd_resp_set_hdr(req, "Content-Disposition", "inline;
filename=capture.bmp");
  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");

  char ts[32];
  snprintf(ts, 32, "%lld.%06ld", fb->timestamp.tv_sec, fb-
>timestamp.tv_usec);
  httpd_resp_set_hdr(req, "X-Timestamp", (const char *)ts);

  uint8_t *buf = NULL;
  size_t buf_len = 0;
  bool converted = frame2bmp(fb, &buf, &buf_len);
  esp_camera_fb_return(fb);
  if (!converted) {
    log_e("BMP Conversion failed");
    httpd_resp_send_500(req);
    return ESP_FAIL;
  }
  res = httpd_resp_send(req, (const char *)buf, buf_len);
  free(buf);
#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
  uint64_t fr_end = esp_timer_get_time();
#endif
  log_i("BMP: %llums, %uB", (uint64_t)((fr_end - fr_start) /
1000), buf_len);
  return res;
}

static size_t jpg_encode_stream(void *arg, size_t index, const
void *data, size_t len) {
  jpg_chunking_t *j = (jpg_chunking_t *)arg;
  if (!index) {
    j->len = 0;
  }
  if (httpd_resp_send_chunk(j->req, (const char *)data, len)
!= ESP_OK) {
    return 0;
  }
  j->len += len;
  return len;
```

```
}

static esp_err_t capture_handler(httpd_req_t *req) {
  camera_fb_t *fb = NULL;
  esp_err_t res = ESP_OK;
#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
  int64_t fr_start = esp_timer_get_time();
#endif

#if CONFIG_LED_ILLUMINATOR_ENABLED
  enable_led(true);
  vTaskDelay(150 / portTICK_PERIOD_MS);  // The LED needs to
be turned on ~150ms before the call to esp_camera_fb_get()
  fb = esp_camera_fb_get();              // or it won't be
visible in the frame. A better way to do this is needed.
  enable_led(false);
#else
  fb = esp_camera_fb_get();
#endif

  if (!fb) {
    log_e("Camera capture failed");
    httpd_resp_send_500(req);
    return ESP_FAIL;
  }

  httpd_resp_set_type(req, "image/jpeg");
  httpd_resp_set_hdr(req, "Content-Disposition", "inline;
filename=capture.jpg");
  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");

  char ts[32];
  snprintf(ts, 32, "%lld.%06ld", fb->timestamp.tv_sec, fb-
>timestamp.tv_usec);
  httpd_resp_set_hdr(req, "X-Timestamp", (const char *)ts);

#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
  size_t fb_len = 0;
#endif
  if (fb->format == PIXFORMAT_JPEG) {
#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
    fb_len = fb->len;
#endif
```

```
    res = httpd_resp_send(req, (const char *)fb->buf, fb-
>len);
  } else {
    jpg_chunking_t jchunk = {req, 0};
    res = frame2jpg_cb(fb, 80, jpg_encode_stream, &jchunk) ?
ESP_OK : ESP_FAIL;
    httpd_resp_send_chunk(req, NULL, 0);
#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
    fb_len = jchunk.len;
#endif
  }
  esp_camera_fb_return(fb);
#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
  int64_t fr_end = esp_timer_get_time();
#endif
  log_i("JPG: %uB %ums", (uint32_t)(fb_len),
(uint32_t)((fr_end - fr_start) / 1000));
  return res;
}

static esp_err_t stream_handler(httpd_req_t *req) {
  camera_fb_t *fb = NULL;
  struct timeval _timestamp;
  esp_err_t res = ESP_OK;
  size_t _jpg_buf_len = 0;
  uint8_t *_jpg_buf = NULL;
  char *part_buf[128];

  static int64_t last_frame = 0;
  if (!last_frame) {
    last_frame = esp_timer_get_time();
  }

  res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
  if (res != ESP_OK) {
    return res;
  }

  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  httpd_resp_set_hdr(req, "X-Framerate", "60");

#if CONFIG_LED_ILLUMINATOR_ENABLED
  isStreaming = true;
  enable_led(true);
```

```
#endif

  while (true) {
    fb = esp_camera_fb_get();
    if (!fb) {
      log_e("Camera capture failed");
      res = ESP_FAIL;
    } else {
      _timestamp.tv_sec = fb->timestamp.tv_sec;
      _timestamp.tv_usec = fb->timestamp.tv_usec;
      if (fb->format != PIXFORMAT_JPEG) {
        bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf,
&_jpg_buf_len);
        esp_camera_fb_return(fb);
        fb = NULL;
        if (!jpeg_converted) {
          log_e("JPEG compression failed");
          res = ESP_FAIL;
        }
      } else {
        _jpg_buf_len = fb->len;
        _jpg_buf = fb->buf;
      }
    }
    if (res == ESP_OK) {
      res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY,
strlen(_STREAM_BOUNDARY));
    }
    if (res == ESP_OK) {
      size_t hlen = snprintf((char *)part_buf, 128,
_STREAM_PART, _jpg_buf_len, _timestamp.tv_sec,
_timestamp.tv_usec);
      res = httpd_resp_send_chunk(req, (const char *)part_buf,
hlen);
    }
    if (res == ESP_OK) {
      res = httpd_resp_send_chunk(req, (const char *)_jpg_buf,
_jpg_buf_len);
    }
    if (fb) {
      esp_camera_fb_return(fb);
      fb = NULL;
      _jpg_buf = NULL;
    } else if (_jpg_buf) {
```

```
      free(_jpg_buf);
      _jpg_buf = NULL;
    }
    if (res != ESP_OK) {
      log_e("Send frame failed");
      break;
    }
    int64_t fr_end = esp_timer_get_time();

    int64_t frame_time = fr_end - last_frame;
    last_frame = fr_end;

    frame_time /= 1000;
#if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
    uint32_t avg_frame_time = ra_filter_run(&ra_filter,
frame_time);
#endif
    log_i(
      "MJPG: %uB %ums (%.1ffps), AVG: %ums (%.1ffps)",
(uint32_t)(_jpg_buf_len), (uint32_t)frame_time, 1000.0 /
(uint32_t)frame_time, avg_frame_time,
      1000.0 / avg_frame_time
    );
  }

#if CONFIG_LED_ILLUMINATOR_ENABLED
  isStreaming = false;
  enable_led(false);
#endif

  return res;
}

static esp_err_t parse_get(httpd_req_t *req, char **obuf) {
  char *buf = NULL;
  size_t buf_len = 0;

  buf_len = httpd_req_get_url_query_len(req) + 1;
  if (buf_len > 1) {
    buf = (char *)malloc(buf_len);
    if (!buf) {
      httpd_resp_send_500(req);
      return ESP_FAIL;
    }
```

```
    if (httpd_req_get_url_query_str(req, buf, buf_len) ==
ESP_OK) {
      *obuf = buf;
      return ESP_OK;
    }
    free(buf);
  }
  httpd_resp_send_404(req);
  return ESP_FAIL;
}

static esp_err_t cmd_handler(httpd_req_t *req) {
  char *buf = NULL;
  char variable[32];
  char value[32];

  if (parse_get(req, &buf) != ESP_OK) {
    return ESP_FAIL;
  }
  if (httpd_query_key_value(buf, "var", variable,
sizeof(variable)) != ESP_OK || httpd_query_key_value(buf,
"val", value, sizeof(value)) != ESP_OK) {
    free(buf);
    httpd_resp_send_404(req);
    return ESP_FAIL;
  }
  free(buf);

  int val = atoi(value);
  log_i("%s = %d", variable, val);
  sensor_t *s = esp_camera_sensor_get();
  int res = 0;

  if (!strcmp(variable, "framesize")) {
    if (s->pixformat == PIXFORMAT_JPEG) {
      res = s->set_framesize(s, (framesize_t)val);
    }
  } else if (!strcmp(variable, "quality")) {
    res = s->set_quality(s, val);
  } else if (!strcmp(variable, "contrast")) {
    res = s->set_contrast(s, val);
  } else if (!strcmp(variable, "brightness")) {
    res = s->set_brightness(s, val);
  } else if (!strcmp(variable, "saturation")) {
```

```
    res = s->set_saturation(s, val);
  } else if (!strcmp(variable, "gainceiling")) {
    res = s->set_gainceiling(s, (gainceiling_t)val);
  } else if (!strcmp(variable, "colorbar")) {
    res = s->set_colorbar(s, val);
  } else if (!strcmp(variable, "awb")) {
    res = s->set_whitebal(s, val);
  } else if (!strcmp(variable, "agc")) {
    res = s->set_gain_ctrl(s, val);
  } else if (!strcmp(variable, "aec")) {
    res = s->set_exposure_ctrl(s, val);
  } else if (!strcmp(variable, "hmirror")) {
    res = s->set_hmirror(s, val);
  } else if (!strcmp(variable, "vflip")) {
    res = s->set_vflip(s, val);
  } else if (!strcmp(variable, "awb_gain")) {
    res = s->set_awb_gain(s, val);
  } else if (!strcmp(variable, "agc_gain")) {
    res = s->set_agc_gain(s, val);
  } else if (!strcmp(variable, "aec_value")) {
    res = s->set_aec_value(s, val);
  } else if (!strcmp(variable, "aec2")) {
    res = s->set_aec2(s, val);
  } else if (!strcmp(variable, "dcw")) {
    res = s->set_dcw(s, val);
  } else if (!strcmp(variable, "bpc")) {
    res = s->set_bpc(s, val);
  } else if (!strcmp(variable, "wpc")) {
    res = s->set_wpc(s, val);
  } else if (!strcmp(variable, "raw_gma")) {
    res = s->set_raw_gma(s, val);
  } else if (!strcmp(variable, "lenc")) {
    res = s->set_lenc(s, val);
  } else if (!strcmp(variable, "special_effect")) {
    res = s->set_special_effect(s, val);
  } else if (!strcmp(variable, "wb_mode")) {
    res = s->set_wb_mode(s, val);
  } else if (!strcmp(variable, "ae_level")) {
    res = s->set_ae_level(s, val);
  }
#if CONFIG_LED_ILLUMINATOR_ENABLED
  else if (!strcmp(variable, "led_intensity")) {
    led_duty = val;
    if (isStreaming) {
```

```
      enable_led(true);
    }
  }
#endif
  else {
    log_i("Unknown command: %s", variable);
    res = -1;
  }

  if (res < 0) {
    return httpd_resp_send_500(req);
  }

  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  return httpd_resp_send(req, NULL, 0);
}

static int print_reg(char *p, sensor_t *s, uint16_t reg,
uint32_t mask) {
  return sprintf(p, "\"0x%x\":%u,", reg, s->get_reg(s, reg,
mask));
}

static esp_err_t status_handler(httpd_req_t *req) {
  static char json_response[1024];

  sensor_t *s = esp_camera_sensor_get();
  char *p = json_response;
  *p++ = '{';

  if (s->id.PID == OV5640_PID || s->id.PID == OV3660_PID) {
    for (int reg = 0x3400; reg < 0x3406; reg += 2) {
      p += print_reg(p, s, reg, 0xFFF);  //12 bit
    }
    p += print_reg(p, s, 0x3406, 0xFF);

    p += print_reg(p, s, 0x3500, 0xFFFF0);  //16 bit
    p += print_reg(p, s, 0x3503, 0xFF);
    p += print_reg(p, s, 0x350a, 0x3FF);   //10 bit
    p += print_reg(p, s, 0x350c, 0xFFFF);  //16 bit

    for (int reg = 0x5480; reg <= 0x5490; reg++) {
      p += print_reg(p, s, reg, 0xFF);
    }
```

```
    for (int reg = 0x5380; reg <= 0x538b; reg++) {
      p += print_reg(p, s, reg, 0xFF);
    }

    for (int reg = 0x5580; reg < 0x558a; reg++) {
      p += print_reg(p, s, reg, 0xFF);
    }
    p += print_reg(p, s, 0x558a, 0x1FF);  //9 bit
  } else if (s->id.PID == OV2640_PID) {
    p += print_reg(p, s, 0xd3, 0xFF);
    p += print_reg(p, s, 0x111, 0xFF);
    p += print_reg(p, s, 0x132, 0xFF);
  }

  p += sprintf(p, "\"xclk\":%u,", s->xclk_freq_hz / 1000000);
  p += sprintf(p, "\"pixformat\":%u,", s->pixformat);
  p += sprintf(p, "\"framesize\":%u,", s->status.framesize);
  p += sprintf(p, "\"quality\":%u,", s->status.quality);
  p += sprintf(p, "\"brightness\":%d,", s->status.brightness);
  p += sprintf(p, "\"contrast\":%d,", s->status.contrast);
  p += sprintf(p, "\"saturation\":%d,", s->status.saturation);
  p += sprintf(p, "\"sharpness\":%d,", s->status.sharpness);
  p += sprintf(p, "\"special_effect\":%u,", s-
>status.special_effect);
  p += sprintf(p, "\"wb_mode\":%u,", s->status.wb_mode);
  p += sprintf(p, "\"awb\":%u,", s->status.awb);
  p += sprintf(p, "\"awb_gain\":%u,", s->status.awb_gain);
  p += sprintf(p, "\"aec\":%u,", s->status.aec);
  p += sprintf(p, "\"aec2\":%u,", s->status.aec2);
  p += sprintf(p, "\"ae_level\":%d,", s->status.ae_level);
  p += sprintf(p, "\"aec_value\":%u,", s->status.aec_value);
  p += sprintf(p, "\"agc\":%u,", s->status.agc);
  p += sprintf(p, "\"agc_gain\":%u,", s->status.agc_gain);
  p += sprintf(p, "\"gainceiling\":%u,", s-
>status.gainceiling);
  p += sprintf(p, "\"bpc\":%u,", s->status.bpc);
  p += sprintf(p, "\"wpc\":%u,", s->status.wpc);
  p += sprintf(p, "\"raw_gma\":%u,", s->status.raw_gma);
  p += sprintf(p, "\"lenc\":%u,", s->status.lenc);
  p += sprintf(p, "\"hmirror\":%u,", s->status.hmirror);
  p += sprintf(p, "\"dcw\":%u,", s->status.dcw);
  p += sprintf(p, "\"colorbar\":%u", s->status.colorbar);
#if CONFIG_LED_ILLUMINATOR_ENABLED
```

```
  p += sprintf(p, ",\"led_intensity\":%u", led_duty);
#else
  p += sprintf(p, ",\"led_intensity\":%d", -1);
#endif
  *p++ = '}';
  *p++ = 0;
  httpd_resp_set_type(req, "application/json");
  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  return httpd_resp_send(req, json_response,
strlen(json_response));
}

static esp_err_t xclk_handler(httpd_req_t *req) {
  char *buf = NULL;
  char _xclk[32];

  if (parse_get(req, &buf) != ESP_OK) {
    return ESP_FAIL;
  }
  if (httpd_query_key_value(buf, "xclk", _xclk, sizeof(_xclk))
!= ESP_OK) {
    free(buf);
    httpd_resp_send_404(req);
    return ESP_FAIL;
  }
  free(buf);

  int xclk = atoi(_xclk);
  log_i("Set XCLK: %d MHz", xclk);

  sensor_t *s = esp_camera_sensor_get();
  int res = s->set_xclk(s, LEDC_TIMER_0, xclk);
  if (res) {
    return httpd_resp_send_500(req);
  }

  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  return httpd_resp_send(req, NULL, 0);
}

static esp_err_t reg_handler(httpd_req_t *req) {
  char *buf = NULL;
  char _reg[32];
  char _mask[32];
```

```
  char _val[32];

  if (parse_get(req, &buf) != ESP_OK) {
    return ESP_FAIL;
  }
  if (httpd_query_key_value(buf, "reg", _reg, sizeof(_reg)) !=
ESP_OK || httpd_query_key_value(buf, "mask", _mask,
sizeof(_mask)) != ESP_OK
      || httpd_query_key_value(buf, "val", _val, sizeof(_val))
!= ESP_OK) {
    free(buf);
    httpd_resp_send_404(req);
    return ESP_FAIL;
  }
  free(buf);

  int reg = atoi(_reg);
  int mask = atoi(_mask);
  int val = atoi(_val);
  log_i("Set Register: reg: 0x%02x, mask: 0x%02x, value:
0x%02x", reg, mask, val);

  sensor_t *s = esp_camera_sensor_get();
  int res = s->set_reg(s, reg, mask, val);
  if (res) {
    return httpd_resp_send_500(req);
  }

  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  return httpd_resp_send(req, NULL, 0);
}

static esp_err_t greg_handler(httpd_req_t *req) {
  char *buf = NULL;
  char _reg[32];
  char _mask[32];

  if (parse_get(req, &buf) != ESP_OK) {
    return ESP_FAIL;
  }
  if (httpd_query_key_value(buf, "reg", _reg, sizeof(_reg)) !=
ESP_OK || httpd_query_key_value(buf, "mask", _mask,
sizeof(_mask)) != ESP_OK) {
    free(buf);
```

```
    httpd_resp_send_404(req);
    return ESP_FAIL;
  }
  free(buf);

  int reg = atoi(_reg);
  int mask = atoi(_mask);
  sensor_t *s = esp_camera_sensor_get();
  int res = s->get_reg(s, reg, mask);
  if (res < 0) {
    return httpd_resp_send_500(req);
  }
  log_i("Get Register: reg: 0x%02x, mask: 0x%02x, value:
0x%02x", reg, mask, res);

  char buffer[20];
  const char *val = itoa(res, buffer, 10);
  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  return httpd_resp_send(req, val, strlen(val));
}

static int parse_get_var(char *buf, const char *key, int def)
{
  char _int[16];
  if (httpd_query_key_value(buf, key, _int, sizeof(_int)) !=
ESP_OK) {
    return def;
  }
  return atoi(_int);
}

static esp_err_t pll_handler(httpd_req_t *req) {
  char *buf = NULL;

  if (parse_get(req, &buf) != ESP_OK) {
    return ESP_FAIL;
  }

  int bypass = parse_get_var(buf, "bypass", 0);
  int mul = parse_get_var(buf, "mul", 0);
  int sys = parse_get_var(buf, "sys", 0);
  int root = parse_get_var(buf, "root", 0);
  int pre = parse_get_var(buf, "pre", 0);
  int seld5 = parse_get_var(buf, "seld5", 0);
```

```
  int pclken = parse_get_var(buf, "pclken", 0);
  int pclk = parse_get_var(buf, "pclk", 0);
  free(buf);

  log_i("Set Pll: bypass: %d, mul: %d, sys: %d, root: %d, pre:
%d, seld5: %d, pclken: %d, pclk: %d", bypass, mul, sys, root,
pre, seld5, pclken, pclk);
  sensor_t *s = esp_camera_sensor_get();
  int res = s->set_pll(s, bypass, mul, sys, root, pre, seld5,
pclken, pclk);
  if (res) {
    return httpd_resp_send_500(req);
  }

  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  return httpd_resp_send(req, NULL, 0);
}

static esp_err_t win_handler(httpd_req_t *req) {
  char *buf = NULL;

  if (parse_get(req, &buf) != ESP_OK) {
    return ESP_FAIL;
  }

  int startX = parse_get_var(buf, "sx", 0);
  int startY = parse_get_var(buf, "sy", 0);
  int endX = parse_get_var(buf, "ex", 0);
  int endY = parse_get_var(buf, "ey", 0);
  int offsetX = parse_get_var(buf, "offx", 0);
  int offsetY = parse_get_var(buf, "offy", 0);
  int totalX = parse_get_var(buf, "tx", 0);
  int totalY = parse_get_var(buf, "ty", 0);  //
codespell:ignore totaly
  int outputX = parse_get_var(buf, "ox", 0);
  int outputY = parse_get_var(buf, "oy", 0);
  bool scale = parse_get_var(buf, "scale", 0) == 1;
  bool binning = parse_get_var(buf, "binning", 0) == 1;
  free(buf);

  log_i(
    "Set Window: Start: %d %d, End: %d %d, Offset: %d %d,
Total: %d %d, Output: %d %d, Scale: %u, Binning: %u", startX,
startY, endX, endY, offsetX, offsetY,
```

```
    totalX, totalY, outputX, outputY, scale, binning  //
codespell:ignore totaly
  );
  sensor_t *s = esp_camera_sensor_get();
  int res = s->set_res_raw(s, startX, startY, endX, endY,
offsetX, offsetY, totalX, totalY, outputX, outputY, scale,
binning);  // codespell:ignore totaly
  if (res) {
    return httpd_resp_send_500(req);
  }

  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  return httpd_resp_send(req, NULL, 0);
}

static esp_err_t index_handler(httpd_req_t *req) {
  httpd_resp_set_type(req, "text/html");
  httpd_resp_set_hdr(req, "Content-Encoding", "gzip");
  sensor_t *s = esp_camera_sensor_get();
  if (s != NULL) {
    if (s->id.PID == OV3660_PID) {
      return httpd_resp_send(req, (const char
*)index_ov3660_html_gz, index_ov3660_html_gz_len);
    } else if (s->id.PID == OV5640_PID) {
      return httpd_resp_send(req, (const char
*)index_ov5640_html_gz, index_ov5640_html_gz_len);
    } else {
      return httpd_resp_send(req, (const char
*)index_ov2640_html_gz, index_ov2640_html_gz_len);
    }
  } else {
    log_e("Camera sensor not found");
    return httpd_resp_send_500(req);
  }
}

void startCameraServer() {
  httpd_config_t config = HTTPD_DEFAULT_CONFIG();
  config.max_uri_handlers = 16;

  httpd_uri_t index_uri = {
    .uri = "/",
    .method = HTTP_GET,
    .handler = index_handler,
```

```
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t status_uri = {
    .uri = "/status",
    .method = HTTP_GET,
    .handler = status_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t cmd_uri = {
    .uri = "/control",
    .method = HTTP_GET,
    .handler = cmd_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t capture_uri = {
    .uri = "/capture",
    .method = HTTP_GET,
    .handler = capture_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
```

```
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t stream_uri = {
    .uri = "/stream",
    .method = HTTP_GET,
    .handler = stream_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t bmp_uri = {
    .uri = "/bmp",
    .method = HTTP_GET,
    .handler = bmp_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t xclk_uri = {
    .uri = "/xclk",
    .method = HTTP_GET,
    .handler = xclk_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t reg_uri = {
```

```
    .uri = "/reg",
    .method = HTTP_GET,
    .handler = reg_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t greg_uri = {
    .uri = "/greg",
    .method = HTTP_GET,
    .handler = greg_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t pll_uri = {
    .uri = "/pll",
    .method = HTTP_GET,
    .handler = pll_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  httpd_uri_t win_uri = {
    .uri = "/resolution",
    .method = HTTP_GET,
    .handler = win_handler,
    .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
```

```
    ,
    .is_websocket = true,
    .handle_ws_control_frames = false,
    .supported_subprotocol = NULL
#endif
  };

  ra_filter_init(&ra_filter, 20);

  log_i("Starting web server on port: '%d'",
config.server_port);
  if (httpd_start(&camera_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(camera_httpd, &index_uri);
    httpd_register_uri_handler(camera_httpd, &cmd_uri);
    httpd_register_uri_handler(camera_httpd, &status_uri);
    httpd_register_uri_handler(camera_httpd, &capture_uri);
    httpd_register_uri_handler(camera_httpd, &bmp_uri);

    httpd_register_uri_handler(camera_httpd, &xclk_uri);
    httpd_register_uri_handler(camera_httpd, &reg_uri);
    httpd_register_uri_handler(camera_httpd, &greg_uri);
    httpd_register_uri_handler(camera_httpd, &pll_uri);
    httpd_register_uri_handler(camera_httpd, &win_uri);
  }

  config.server_port += 1;
  config.ctrl_port += 1;
  log_i("Starting stream server on port: '%d'",
config.server_port);
  if (httpd_start(&stream_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(stream_httpd, &stream_uri);
  }
}

void setupLedFlash(int pin) {
#if CONFIG_LED_ILLUMINATOR_ENABLED
  ledcAttach(pin, 5000, 8);
#else
  log_i("LED flash is disabled ->
CONFIG_LED_ILLUMINATOR_ENABLED = 0");
#endif
}
```

camera_index.h:

```
#define index_ov3660_html_gz_len 8636
const unsigned char index_ov3660_html_gz[] = {
  0x1F, 0x8B, 0x08, 0x08, 0xD3, 0xA3, 0x7B, 0x67, 0x00, 0x03,
0x69, 0x6E, 0x64, 0x65, 0x78, 0x5F, 0x6F, 0x76, 0x33, 0x36,
0x36, 0x30, 0x2E, 0x68, 0x74, 0x6D,
  0x6C, 0x00, 0xED, 0x3D, 0x69, 0x73, 0xDB, 0x46, 0xB2, 0xDF,
0xFD, 0x2B, 0x60, 0x66, 0xD7, 0xA2, 0xCA, 0x22, 0x45, 0xF0,
0xD2, 0x61, 0x89, 0x7E, 0xB6, 0xAC,
  0xD8, 0xA9, 0xB5, 0xB3, 0xDE, 0x28, 0x71, 0x92, 0xDA, 0xDA,
0x72, 0x40, 0x62, 0x48, 0x22, 0x06, 0x01, 0x2E, 0x00, 0xEA,
0x58, 0x97, 0x7E, 0xC7, 0xFB, 0x41,
  0xEF, 0x8F, 0xBD, 0xEE, 0x39, 0x70, 0x71, 0x00, 0x0C, 0x00,
0x92, 0x52, 0xF2, 0x1E, 0x5D, 0x65, 0xE1, 0x98, 0xEE, 0xE9,
0x7B, 0x7A, 0x7A, 0x06, 0xC0, 0xD9,
  0x53, 0xD3, 0x9D, 0x04, 0x77, 0x4B, 0xA2, 0xCD, 0x83, 0x85,
0x3D, 0x7A, 0x72, 0xC6, 0xFE, 0x68, 0xF0, 0x3B, 0x9B, 0x13,
0xC3, 0x64, 0x87, 0xF4, 0x74, 0x41,
  0x02, 0x43, 0x9B, 0xCC, 0x0D, 0xCF, 0x27, 0xC1, 0x79, 0x63,
0x15, 0x4C, 0x5B, 0xC7, 0x8D, 0xF4, 0x6D, 0xC7, 0x58, 0x90,
0xF3, 0xC6, 0xB5, 0x45, 0x6E, 0x96,
  0xAE, 0x17, 0x34, 0xB4, 0x89, 0xEB, 0x04, 0xC4, 0x81, 0xE6,
0x37, 0x96, 0x19, 0xCC, 0xCF, 0x4D, 0x72, 0x6D, 0x4D, 0x48,
0x8B, 0x9E, 0x1C, 0x58, 0x8E, 0x15,
  0x58, 0x86, 0xDD, 0xF2, 0x27, 0x86, 0x4D, 0xCE, 0xF5, 0x38,
0xAE, 0xC0, 0x0A, 0x6C, 0x32, 0xBA, 0xBC, 0xFA, 0xD8, 0xEB,
0x6A, 0x7F, 0xFF, 0xD4, 0x1B, 0x0E,
  0x3B, 0x67, 0x87, 0xEC, 0x5A, 0xD4, 0xC6, 0x0F, 0xEE, 0xE2,
0xE7, 0xF8, 0x1B, 0xBB, 0xE6, 0x9D, 0xF6, 0x35, 0x71, 0x09,
0x7F, 0x53, 0x20, 0xA2, 0x35, 0x35,
  0x16, 0x96, 0x7D, 0x77, 0xAA, 0xBD, 0xF2, 0xA0, 0xCF, 0x83,
0x77, 0xC4, 0xBE, 0x26, 0x81, 0x35, 0x31, 0x0E, 0x7C, 0xC3,
0xF1, 0x5B, 0x3E, 0xF1, 0xAC, 0xE9,
  0x8B, 0x35, 0xC0, 0xB1, 0x31, 0xF9, 0x32, 0xF3, 0xDC, 0x95,
0x63, 0x9E, 0x6A, 0xDF, 0xE8, 0xC7, 0xF8, 0x6F, 0xBD, 0xD1,
0xC4, 0xB5, 0x5D, 0x0F, 0xEE, 0x5F,
  0x7E, 0x8B, 0xFF, 0xD6, 0xEF, 0xD3, 0xDE, 0x7D, 0xEB, 0x3F,
0xE4, 0x54, 0xD3, 0x87, 0xCB, 0xDB, 0xC4, 0xFD, 0xFB, 0x27,
0x89, 0xD3, 0x79, 0x37, 0x8B, 0x7A,
  0x0E, 0x7F, 0x9C, 0x0F, 0xEF, 0x93, 0x49, 0x60, 0xB9, 0x4E,
0x7B, 0x61, 0x58, 0x8E, 0x04, 0x93, 0x69, 0xF9, 0x4B, 0xDB,
0x00, 0x19, 0x4C, 0x6D, 0x92, 0x8B,
```

```
  0xE7, 0x9B, 0x05, 0x71, 0x56, 0x07, 0x05, 0xD8, 0x10, 0x49,
0xCB, 0xB4, 0x3C, 0xD6, 0xEA, 0x14, 0xE5, 0xB0, 0x5A, 0x38,
0x85, 0x68, 0xF3, 0xE8, 0x72, 0x5C,
  0x87, 0x48, 0x04, 0x88, 0x1D, 0xDD, 0x78, 0xC6, 0x12, 0x1B,
0xE0, 0xDF, 0xF5, 0x26, 0x0B, 0xCB, 0x61, 0x46, 0x75, 0xAA,
0xF5, 0xFA, 0x9D, 0xE5, 0x6D, 0x81,
  0x2A, 0x7B, 0x43, 0xFC, 0xB7, 0xDE, 0x68, 0x69, 0x98, 0xA6,
0xE5, 0xCC, 0x4E, 0xB5, 0x63, 0x29, 0x0A, 0xD7, 0x33, 0x89,
0xD7, 0xF2, 0x0C, 0xD3, 0x5A, 0xF9,
  0xA7, 0x5A, 0x5F, 0xD6, 0x66, 0x61, 0x78, 0x33, 0xA0, 0x25,
0x70, 0x81, 0xD8, 0x96, 0x2E, 0xA5, 0x84, 0x37, 0xF1, 0xAC,
0xD9, 0x3C, 0x00, 0x95, 0xAE, 0xB5,
  0x49, 0x0B, 0x8D, 0xBB, 0x50, 0x91, 0x3E, 0x73, 0xE5, 0x26,
0x97, 0x9A, 0x61, 0x5B, 0x33, 0xA7, 0x65, 0x05, 0x64, 0x01,
0xEC, 0xF8, 0x81, 0x47, 0x82, 0xC9,
  0x3C, 0x8F, 0x94, 0xA9, 0x35, 0x5B, 0x79, 0x44, 0x42, 0x48,
0x28, 0xB7, 0x1C,
  0xF8, 0x30, 0xCA, 0xE2, 0xDC, 0x3E, 0xB4, 0xAA, 0xC6, 0xCB,
0xF2, 0xE1, 0x10, 0x60, 0x1A, 0xA3, 0xD7, 0x1F, 0x77, 0x93,
0xFB, 0x61, 0x67, 0x8A, 0x1A, 0xAA,
  0xA5, 0x0F, 0xCA, 0xD4, 0x43, 0x6B, 0xE3, 0xA6, 0x82, 0x36,
0x6E, 0x90, 0xF0, 0x9F, 0x77, 0xA4, 0x8D, 0x1B, 0x75, 0x6D,
0x6C, 0xD8, 0x5F, 0x6E, 0x1E, 0x83,
  0x7E, 0xE8, 0x43, 0x87, 0x63, 0xA3, 0xFC, 0x70, 0x24, 0x00,
0x71, 0xD3, 0x18, 0x1C, 0x69, 0xAF, 0x8D, 0xDD, 0x0C, 0x48,
0x61, 0xBF, 0xBB, 0x70, 0xA1, 0x88,
  0xC9, 0x87, 0xD6, 0x93, 0x4D, 0xCC, 0x0A, 0x49, 0x9E, 0xF9,
0x19, 0x9F, 0xE0, 0xC3, 0x27, 0xDB, 0xEF, 0x20, 0xDB, 0xBB,
0x7C, 0xA3, 0x7D, 0x27, 0x4E, 0x1F,
  0xAA, 0x30, 0x94, 0xA4, 0x29, 0x39, 0x6F, 0xEA, 0x0E, 0xB6,
0xB5, 0x2D, 0x03, 0x30, 0xD7, 0xD4, 0x4D, 0xFC, 0x21, 0x30,
0xFE, 0xFA, 0xD4, 0x22, 0x52, 0xF8,
  0xC3, 0x48, 0x74, 0xBB, 0x03, 0x09, 0x5A, 0x7E, 0x60, 0xD9,
0x36, 0x4C, 0x92, 0x48, 0xA0, 0x5D, 0xE1, 0xA1, 0xE2, 0xD3,
0x47, 0x31, 0x2C, 0xE2, 0xD9, 0xC3,
  0xC0, 0x23, 0xC6, 0xA2, 0x31, 0xBA, 0xC2, 0x17, 0xCB, 0x02,
0x2E, 0x3C, 0x2B, 0x46, 0x16, 0x7F, 0x4E, 0x29, 0xDF, 0x04,
0xE9, 0x93, 0x89, 0xF8, 0xA8, 0x61,
  0xF2, 0x45, 0xD0, 0xE0, 0x03, 0xEC, 0xC1, 0xE3, 0xD1, 0x99,
0xBF, 0x34, 0x1C, 0xD1, 0x8C, 0x3E, 0x95, 0x7B, 0xC3, 0x1F,
0xB3, 0x1C, 0xBB, 0xB6, 0xF9, 0x22,
  0xB6, 0x12, 0x78, 0x15, 0x3E, 0x2F, 0x88, 0x20, 0xE0, 0x44,
0x02, 0x43, 0x81, 0xB4, 0xE7, 0x9E, 0x40, 0xCF, 0x1E, 0xED,
0xC4, 0xD7, 0x08, 0xE5, 0x88, 0x3B,
```

```
  0xE3, 0x11, 0x47, 0x8F, 0xCC, 0x42, 0x2B, 0x92, 0x3D, 0xFA,
0x2A, 0x7D, 0xE0, 0xF1, 0x07, 0x32, 0xB3, 0x7C, 0xA0, 0x51,
0x03, 0x45, 0x1D, 0xD2, 0x87, 0xC4,
  0x98, 0xA3, 0xA8, 0x3D, 0x80, 0x18, 0xEF, 0x92, 0x3F, 0x3F,
0x2D, 0x7D, 0xAE, 0xB4, 0xD4, 0x58, 0x92, 0x7E, 0x08, 0x34,
0x89, 0xB1, 0xC8, 0x0C, 0x9F, 0xB6,
  0x5A, 0xF3, 0x3E, 0x3E, 0xEE, 0xA6, 0x09, 0xD6, 0xCE, 0x0E,
0xE7, 0xFD, 0xA2, 0xC7, 0x89, 0x0A, 0x9F, 0x55, 0x04, 0x4E,
0x2B, 0x3F, 0xAA, 0x88, 0x52, 0x1A,
  0x01, 0x35, 0x07, 0xDA, 0x07, 0xC3, 0xFF, 0x72, 0xA0, 0x7D,
0x42, 0x9F, 0xDF, 0xE1, 0x13, 0x8B, 0x48, 0xBB, 0x61, 0x9A,
0x5E, 0xE6, 0x53, 0x8B, 0xFD, 0xC4,
  0x53, 0x8B, 0x43, 0xF1, 0xD4, 0x62, 0x54, 0xBA, 0xEE, 0xDC,
0xF6, 0x3A, 0x9D, 0x63, 0x15, 0xD6, 0x15, 0x9F, 0x5C, 0xDC,
0x08, 0x4F, 0x0B, 0x90, 0xA6, 0x22,
  0x4F, 0x7D, 0xC1, 0x53, 0x6C, 0x07, 0xEF, 0xED, 0x74, 0xFA,
0xD8, 0x38, 0xE2, 0x6B, 0x08, 0xD5, 0x59, 0xEA, 0x74, 0x77,
0xFD, 0x78, 0x29, 0x35, 0xEE, 0x4D,
  0x3D, 0x5D, 0x4A, 0x9B, 0xA4, 0xA3, 0xE1, 0x20, 0x37, 0x18,
0x52, 0x10, 0xE6, 0xF4, 0x6F, 0x37, 0xE9, 0xF4, 0xB3, 0x1A,
0x4E, 0x3F, 0x5B, 0x73, 0xFA, 0x1D,
  0x7A, 0xBB, 0x20, 0xFC, 0xCF, 0xE6, 0xF1, 0x82, 0xAF, 0x12,
0x5E, 0x2F, 0xE5, 0xAB, 0xD3, 0xD9, 0xA8, 0xDF, 0x17, 0x3A,
0x49, 0x68, 0x0C, 0x6F, 0x37, 0xE9,
  0x24, 0x19, 0xA6, 0x5B, 0xC9, 0x4E, 0x79, 0xD8, 0x19, 0xED,
0x66, 0x5C, 0xA2, 0xD9, 0x54, 0x5C, 0xA1, 0xBC, 0x77, 0x7C,
0x7E, 0xAF, 0xD7, 0xE7, 0xA9, 0xD3,
  0x26, 0xD4, 0xA3, 0xFE, 0x24, 0x7B, 0x66, 0x93, 0xCD, 0x24,
0x66, 0x4B, 0x48, 0x84, 0x4B, 0x27, 0x66, 0x1F, 0xDF, 0xBF,
0x2F, 0x97, 0x8B, 0xC5, 0x7B, 0x79,
  0x24, 0xB9, 0x58, 0x6E, 0xDD, 0xEA, 0x6E, 0x09, 0x37, 0x90,
0xEA, 0x4A, 0xA6, 0x1B, 0x81, 0x37, 0x46, 0xAF, 0xE9, 0xB1,
0x16, 0x93, 0x58, 0x29, 0xE3, 0x55,
  0x9E, 0x96, 0x53, 0xC0, 0x58, 0x61, 0x2B, 0x22, 0x21, 0xAD,
0x1B, 0x45, 0x5C, 0x39, 0xC5, 0xAC, 0x18, 0x7B, 0xEA, 0x4C,
0xD5, 0xF6, 0x09, 0xDA, 0xA4, 0x28,
  0x15, 0x5E, 0xAC, 0xEC, 0xCA, 0x6A, 0xE3, 0xB0, 0x8D, 0xD1,
0x07, 0x98, 0xE3, 0x5A, 0x4B, 0xDB, 0x82, 0x99, 0x47, 0xB3,
0xA3, 0xB5, 0xB4, 0x9E, 0xBE, 0xBF,
  0xC3, 0x31, 0x52, 0x90, 0x51, 0xF2, 0x35, 0x1E, 0x7A, 0xF4,
0x74, 0x4B, 0x6F, 0x43, 0xEF, 0xF1, 0xA8, 0xAB, 0x10, 0xCF,
0x75, 0x83, 0xCA, 0xDA, 0x10, 0xC0,
```

```
  0x90, 0xA8, 0xC0, 0x91, 0x16, 0xE9, 0x44, 0x5D, 0x15, 0xB1,
0xBD, 0x76, 0x11, 0x36, 0x35, 0x75, 0x28, 0xED, 0xAC, 0xC3,
0x05, 0x6B, 0xD5, 0x2D, 0x69, 0x12,
  0xAC, 0x7A, 0x63, 0xD4, 0x2D, 0x81, 0xA1, 0x78, 0x63, 0x1A,
0x6B, 0x55, 0xDF, 0x89, 0xFC, 0xBB, 0xEA, 0xB1, 0x8F, 0xC3,
0x42, 0xDA, 0x7D, 0x07, 0xA9, 0xEE,
  };
```

ci.json:

```
{
  "fqbn": {
    "esp32s3": [
      "espressif:esp32:esp32s3:PSRAM=opi,USBMode=default,Parti
tionScheme=custom,FlashMode=qio",
      "espressif:esp32:esp32s3:PSRAM=enabled,USBMode=default,P
artitionScheme=custom,FlashMode=qio",
      "espressif:esp32:esp32s3:PSRAM=disabled,USBMode=default,
PartitionScheme=custom,FlashMode=qio"
    ]
  }
  "requires": [
    "CONFIG_CAMERA_TASK_STACK_SIZE=[0-9]+"
  ]
}
```

Лістинг 3.1 – Програмний код розроблено для мікроконтролера ESP32-S3 у середовищі Arduino IDE