Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп’ютерної інженерії та управління
(повна назва)
Кафедра Комп'ютерних інтелектуальних технологій та систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Розробка мікросервісної системи з підтримкою паралельної обробки запитів для аналізу телеметричних даних в реальному часі

Виконав:
здобувач IV року навчання,
групи КІУКІ-21-10
Євгеній МАСЛОВ
(власне ім'я, прізвище)
Спеціальність 123 Комп’ютерна інженерія
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Комп'ютерна інженерія
(повна назва освітньої програми)
Керівник Ст. викладач Максим КУШНАРЬОВ
(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри Олег РУДЕНКО

2025 р.

Харківський національний університет радіоелектроніки

Факультет	Комп'ютерної інженерії та управління
Кафедра	Комп'ютерних інтелектуальних технологій та систем
Рівень вищої освіти	перший (бакалаврський)
Спеціальність	123 Комп'ютерна інженерія
Тип програми	освітньо-професійна
Освітня програма	Комп'ютерна інженерія

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____»_____2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Маслову Євгенію Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка мікросервісної системи з підтримкою паралельної обробки запитів для аналізу телеметричних даних в реальному часі

затверджена наказом по університету від " 21 " травня 2025 р. № 399СТ

2. Термін подання здобувачем роботи до екзаменаційної комісії 14.06.2025

3. Вхідні дані до роботи Дані з сенсорних вузлів на базі ESP32 та Arduino Nano
Протоколи передачі телеметрії (MQTT, UART, LoRa);
Потокові дані, зібрані у тепличних умовах та лабораторних експериментах;
Конфігурації мікросервісної системи для збору, обробки й зберігання часових рядів
Алгоритмічні моделі виявлення критичних ситуацій
Статистичні метрики для оцінки затримки, точності та енергоефективності вузлі

4. Перелік питань, що потрібно опрацювати в роботі
Аналіз предметної області та існуючих підходів до обробки телеметричних даних
Проектування архітектури мікросервісної системи збору, обробки та візуалізації
Порівняльне дослідження вузлів на основі ESP32 та Arduino Nano
Розробка й реалізація алгоритму виявлення критичних ситуацій
Інтеграція мікросервісів за допомогою REST API/gRPC
Аналіз результатів, визначення основних переваг та недоліків розглянутих платформ

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) _____
10 слайдів

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд і аналіз сучасного стану розглянутої проблеми, а також існуючих методів і засобів вирішення задач кваліфікаційної роботи	26.05.2025 – 31.05.2025	Виконано
2	Проектування архітектури додатку	01.06.2025 – 03.06.2025	Виконано
3	Створення додатку	04.06.2025 – 07.06.2025	Виконано
4	Аналіз та налагодження роботи додатку	08.06.2025 – 09.06.2025	Виконано
5	Оформлення матеріалів кваліфікаційної роботи	10.06.2025 – 14.06.2025	Виконано

Дата видачі завдання 26.05.2025

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Максим КУШНАРЬОВ

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 70 с., 13 рис., 8 табл., 1 дод., 18 джерел.

МІКРОСЕРВІСИ, ТЕЛЕМЕТРІЯ, IOT, РОЗПОДІЛЕНА СИСТЕМА, ХМАРНІ ОБЧИСЛЕННЯ, ПАРАЛЕЛЬНА ОБРОБКА

У роботі розроблено та експериментально досліджено мікросервісну систему збору, обробки та аналізу телеметричних даних у реальному часі на базі IoT-вузлів ESP32 та Arduino Nano. Запропоновано архітектуру розподіленої системи з підтримкою паралельної обробки потоків, використанням сучасних протоколів передачі (MQTT, UART, LoRa) та оптимізованих сховищ (InfluxDB). Особливу увагу приділено реалізації паралельної обробки потоків: телеметричні дані буферизуються у Kafka, а потім паралельно аналізуються у Flink, що дозволяє оперативно виявляти критичні ситуації, наприклад, низьку вологість. Проведено тестування й порівняння вузлів за затримкою, точністю, енергоспоживанням, що дозволило визначити доцільність застосування кожної платформи для задач агротелеметрії. Розглянуто інструменти візуалізації та автоматизації керування (Grafana, Node-RED). Отримані результати дозволяють оптимізувати процеси моніторингу та автоматичного реагування у сучасних агросистемах.

ABSTRACT

Explanatory note of qualification work 70 pages, 13 figures, 8 tables, 1 appendices, 18 sources.

MICROSERVISES, TELEMETRY, IOT, DISTRIBUTED SYSTEM, CLOUD COMPUTING, PARALLEL PROCESSING

The paper presents the development and experimental evaluation of a microservices-based system for real-time collection, processing, and analysis of telemetry data using IoT nodes based on ESP32 and Arduino Nano. The architecture supports distributed data flows, parallel processing, and employs modern transmission protocols (MQTT, UART, LoRa) and time series databases (InfluxDB). Special attention is paid to the realization of parallel data processing: telemetry streams are buffered in Kafka and then processed in parallel by Flink, enabling prompt detection of critical situations such as low humidity. The system was tested and the nodes were compared in terms of latency, accuracy, and energy efficiency, enabling a justified choice of platform for agri-telemetry tasks. Visualization and automation tools (Grafana, Node-RED) are discussed. The results provide recommendations for optimizing monitoring processes and automated response in modern agrotechnical systems.

ЗМІСТ

омилка! Закладку не визначено.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

IoT– Internet of Things (Інтернет речей)

MQTT – Message Queuing Telemetry Transport

REST – Representational State Transfer

gRPC – Google Remote Procedure Call

API – Application Programming Interface

DB – Database (база даних)

MAE – Mean Absolute Error (середня абсолютна похибка)

RH – Relative Humidity (відносна вологість)

OTA – Over-The-Air (оновлення по повітрю)

DHT22 – Digital Humidity and Temperature Sensor

LCD – Liquid Crystal Display

CSV – Comma-Separated Values (формат даних)

NTP – Network Time Protocol

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій та зростання обсягів даних дедалі більшої актуальності набувають системи, здатні забезпечувати ефективний збір, обробку та аналіз великих потоків телеметричної інформації в реальному часі. Такий підхід особливо важливий для галузей, де оперативність прийняття рішень залежить від своєчасної обробки телеметричних даних — наприклад, у промисловості, транспорті, енергетиці, телекомунікаціях і моніторингових системах Інтернету речей (IoT).

Традиційні монолітні рішення часто не справляються із зростаючими вимогами до масштабованості, відмовостійкості та гнучкості. У зв’язку з цим усе більшого поширення набуває мікросервісна архітектура, яка дозволяє будувати розподілені системи, здатні масштабуватися горизонтально та забезпечувати безперервну обробку великої кількості запитів. Одним із ключових викликів для таких систем є організація паралельної обробки запитів, що дозволяє значно підвищити продуктивність і скоротити час реагування на критичні події.

Актуальність теми роботи полягає у необхідності створення ефективних, масштабованих та надійних рішень для аналізу телеметричних даних у реальному часі із застосуванням сучасних підходів до розробки програмного забезпечення. Реалізація мікросервісної системи з підтримкою паралельної обробки запитів дозволяє значно підвищити якість обслуговування, знизити затримки та забезпечити гнучкість при інтеграції з іншими інформаційними системами.

.

1 АКТУАЛЬНІСТЬ ЗАДАЧІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

У сучасних умовах цифрової трансформації зростає потреба в обробці телеметричних даних, які надходять у режимі реального часу з численних джерел — сенсорів, пристроїв IoT, транспортних засобів, енергетичних мереж тощо. Такі дані мають ключове значення для забезпечення надійного моніторингу, своєчасного виявлення аномалій, прогнозування відмов і оперативного реагування.

Серед головних викликів у цій сфері — стрімке зростання обсягу інформації, необхідність її аналізу без затримок, гнучке масштабування системи при збільшенні кількості пристроїв, а також потреба в автоматизованому прийнятті рішень. Традиційні монолітні архітектури виявляються недостатньо ефективними для таких завдань, тому все частіше застосовуються мікросервісні підходи, які забезпечують незалежність компонентів, зручність розгортання й масштабування, а також просту інтеграцію нових функцій.

У центрі цієї предметної області — телеметричний пристрій, який передає потік вимірювань до спеціалізованих сервісів обробки. У процесі аналізу можуть виявлятися відхилення від норми, що генерують події чи сповіщення для оперативного реагування. Дані при цьому зберігаються в архівах, що забезпечує подальший доступ до аналітики.

Типовими сценаріями є підключення нових пристроїв, надсилання телеметрії через стандартні протоколи (наприклад, MQTT або Kafka), виявлення критичних ситуацій, формування звітів і візуалізація поточної інформації в режимі реального часу. Система повинна працювати безперервно, забезпечуючи надійну обробку й доставку повідомлень, адаптивність до змін навантаження, інтеграцію з іншими сервісами, а також захист переданих даних.

З урахуванням вимог, архітектура мікросервісної системи включає окремі компоненти для реєстрації пристроїв, прийому потоків даних, їх обробки та

збереження, виявлення аномалій, надсилання сповіщень і візуалізації. Такий поділ дозволяє масштабувати систему вибірково, розгортати сервіси незалежно та адаптувати аналітичну логіку до конкретних потреб.

Для реалізації системи доцільно використовувати стрімінгові платформи (Kafka, RabbitMQ), інструменти обробки потоків (Flink, Spark Streaming), сучасні сховища даних (InfluxDB, TimescaleDB, Elasticsearch), а також засоби візуалізації (Grafana, Kibana). Мікросервіси реалізуються зазвичай за допомогою мов Python, Node.js або Go і розгортаються у контейнерах (Docker), керованих системами на кшталт Kubernetes.

Аналіз телеметричних даних відбувається поетапно: спочатку здійснюється збір та передача показників, далі — фільтрація, нормалізація та агрегація. Після цього дані піддаються аналітичній обробці із застосуванням порогових правил, методів машинного навчання або статистики. Збереження здійснюється як у вигляді «сирих» потоків, так і у вигляді структурованих зведень. Результати можуть візуалізуватись у режимі реального часу або ініціювати автоматизовані дії.

Прикладом може слугувати система моніторингу температури серверів. Кожні кілька секунд сенсори фіксують температуру, дані передаються через MQTT до брокера, обробляються засобами потокової аналітики (наприклад, Spark Streaming), а у разі перевищення критичного значення система надсилає сповіщення адміністратору та фіксує інцидент у базі даних. Усе це — із мінімальною затримкою, що забезпечує оперативність прийняття рішень.

Таким чином, мікросервісна система аналізу телеметричних даних у реальному часі забезпечує надійність, масштабованість і адаптивність сучасних інформаційних рішень, критично важливих для розумної інфраструктури, промислових процесів та цифрового моніторингу.

1.2 Актуальність обраної теми

У сучасних умовах стрімкого зростання обсягів телеметричної інформації, що надходить з численних пристроїв Інтернету речей, промислових датчиків,

транспортних систем і енергетичних мереж, виникає потреба у створенні високопродуктивних рішень для обробки цих даних у реальному часі. Традиційні монолітні підходи до побудови інформаційних систем не здатні забезпечити необхідну швидкодію, масштабованість і надійність в умовах інтенсивного надходження потоків даних.

У зв’язку з цим дедалі більшої актуальності набувають мікросервісні архітектури, які забезпечують гнучке розгортання, незалежність компонентів і простоту масштабування. Особливо важливим є впровадження паралельної обробки запитів, що дозволяє суттєво підвищити продуктивність системи та зменшити затримки при аналізі телеметричних потоків.

Розробка мікросервісної системи з підтримкою паралельної обробки запитів спрямована на вирішення нагальних завдань, пов’язаних із забезпеченням безпеки, надійності й ефективності критичних інфраструктур — від автоматизованих систем управління виробництвом до інтелектуального моніторингу в міському середовищі. Актуальність теми також підтверджується запитами IT-індустрії на створення масштабованих, стійких до збоїв рішень, здатних обробляти великі обсяги даних у реальному часі, що є основою цифровізації сучасного світу.

1.3 Огляд існуючих рішень

Аналіз телеметричних даних у реальному часі є однією з ключових задач в системах моніторингу, керування та прогнозування. За останні роки в цій сфері з’явився ряд технічних рішень, що поєднують потокову обробку даних, мікросервісну архітектуру та масштабованість за рахунок паралельних обчислень. Розглянемо основні підходи та реалізації.

1.3.1 Хмарні платформи потокової аналітики

Kafka — одна з найпоширеніших платформ для збору й обробки телеметричних потоків у реальному часі. Завдяки розподіленій архітектурі, вона дозволяє

масштабувати обробку запитів горизонтально. Використання Kafka Streams або ksqlDB дозволяє безпосередньо у потоковому режимі виконувати фільтрацію, агрегацію, виявлення аномалій тощо [1, 2]. Ці сервіси легко інтегруються в мікросервісні системи.

Flink — це рушій потокової обробки з підтримкою подій з низькою затримкою та багатопоточністю. Перевагою Flink є точна обробка (exactly-once semantics) і підтримка складних сценаріїв обробки. Він активно використовується в промислових системах, де критичним є контроль за телеметрією (наприклад, у Siemens, Uber) [12].

Google Cloud Dataflow (Apache Beam) – хмарне рішення, яке дозволяє реалізовувати як пакетну, так і потокову обробку даних з можливістю абстрагування від конкретної інфраструктури. Має підтримку паралельного виконання за допомогою розподілених обчислень [4].

1.3.2 Мікросервісні платформи для реального часу

Prometheus + Alertmanager + Grafana – популярне рішення для телеметричного моніторингу в DevOps. Дані агрегуються Prometheus-сервісами, обробляються та виводяться на Grafana-дешбордах. Паралельність забезпечується через незалежну обробку запитів кожним сервісом, що працює як мікросервіс [5].

ThingsBoard – Open-source платформа IoT, що має вбудовану підтримку потокової обробки телеметрії, правила обробки подій та мікросервісну структуру. Працює з MQTT, CoAP, HTTP та підтримує Docker/Kubernetes для масштабування [6, 7]. Azure IoT Hub + Stream Analytics – комбінує прийом телеметрії від пристроїв, обробку запитів у режимі реального часу й інтеграцію з іншими Azure-сервісами. Паралельна обробка реалізується через автоматичне масштабування потоків [8].

1.3.3 Промислові рішення

Uber’s M3 (Metrics Platform) – система збору та аналізу мільярдів метрик за секунду. В основі — мікросервісна архітектура, що включає паралельні обробники

(query engine, ingest engine) та високопродуктивне збереження даних. M3 використовується для моніторингу інфраструктури й сервісів [9].

Tesla Telemetry System – система телеметрії Tesla обробляє великі обсяги даних з автопарку в реальному часі. Застосовується мікросервісна архітектура з брокерами повідомлень, контейнеризацією та хмарною обробкою даних, включаючи моделі машинного навчання.

1.3.4 Технологічні патерни і стандарти

CQRS (Command Query Responsibility Segregation) — поділ запитів на читання та запис дозволяє обробляти їх паралельно й ефективно масштабувати.

Event-Driven Architecture (EDA) — дозволяє будувати реактивні системи, де сервіси взаємодіють через події.

Reactive Streams (Spring WebFlux, Akka Streams) — підтримують неблокуючу, асинхронну обробку даних у багатьох мовах програмування [10, 11].

Існуючі рішення демонструють ефективність комбінації мікросервісного підходу, потокової аналітики та паралельної обробки для побудови систем аналізу телеметричних даних у реальному часі. Основні переваги таких систем — масштабованість, гнучкість і здатність реагувати на події майже миттєво. Це робить подібні рішення особливо цінними у сферах, де критична кожна секунда: транспорт, енергетика, виробництво, безпека. Однак вибір платформи має базуватися на конкретних потребах: очікуване навантаження, тип даних, вимоги до затримок і доступності.

1.3.5 Наукові та інноваційні підходи до інтелектуального контролю мікроклімату

Окрему увагу заслуговують сучасні наукові розробки, що поєднують мікросервісні підходи, IoT та алгоритми штучного інтелекту для задач автоматизованого моніторингу та керування мікрокліматом.

Зокрема, у роботі [17] розглянуто застосування Q-learning — алгоритму підсиленого навчання — для оптимізації мікроклімату у міських фермах. Запропонована система об'єднує телеметричний збір даних із сенсорів, мікросервісну архітектуру обробки та інтелектуальний агент керування. Дослідження показало, що використання адаптивного Q-learning забезпечує підвищення врожайності за рахунок оптимального поєднання режимів поливу, вентиляції та обігріву.

У роботі [18] детально описано архітектуру програмно-апаратної системи моніторингу і керування мікрокліматом у тепличних або закритих приміщеннях. Особливістю підходу є модульність рішень, розподілене збирання телеметрії та інтеграція інтелектуальних сервісів аналізу даних для формування оптимальних стратегій впливу на середовище.

Таким чином, поєднання мікросервісних платформ із інтелектуальними алгоритмами керування стає провідним напрямом у розробці сучасних систем агромоніторингу, що підтверджують і результати наукових досліджень у даній галузі.

1.4 Постановка задачі

У зв’язку зі стрімким зростанням кількості телеметричних пристроїв та обсягів даних, які вони генерують, зростає потреба у створенні масштабованих інформаційних систем, здатних ефективно обробляти телеметричні потоки в реальному часі. Традиційні монолітні підходи не забезпечують необхідної продуктивності, гнучкості та надійності в умовах високого навантаження та розподіленого середовища.

Актуальним є впровадження мікросервісної архітектури, яка дозволяє реалізувати паралельну обробку запитів, розподіл функціональних компонентів, динамічне масштабування, а також безперервну інтеграцію нових сервісів. Особливо важливим є забезпечення низької затримки, виявлення аномалій та виведення результатів у режимі реального часу.

Метою даної бакалаврської роботи є розробка мікросервісної програмної системи для збору, обробки, аналізу та візуалізації телеметричних даних, що надходять з віддалених пристроїв, із підтримкою паралельної обробки запитів.

Для досягнення мети необхідно вирішити такі основні задачі:

– Проаналізувати предметну область та існуючі підходи до обробки телеметричних даних у режимі реального часу, з урахуванням вимог до масштабованості та швидкодії.

– Спроєктувати архітектуру системи на основі мікросервісного підходу з виділенням основних сервісів: реєстрації пристроїв, збору даних, обробки потоків, виявлення аномалій, сповіщення та візуалізації.

– Реалізувати підтримку паралельної обробки за допомогою відповідних технологій потокової обробки даних (Apache Kafka, Apache Flink).

– Забезпечити взаємодію між мікросервісами через брокер повідомлень або REST/gRPC API.

– Оцінити продуктивність системи, її масштабованість та здатність до виявлення критичних ситуацій з мінімальною затримкою.

Результатом роботи має стати прототип мікросервісної системи, який демонструє ефективну обробку телеметричних потоків із можливістю масштабування та подальшого розширення функціональності.

2 МОДЕЛЬ МІКРОСЕРВІСНОЇ СИСТЕМИ ДЛЯ АНАЛІЗУ ТЕЛЕМЕТРИЧНИХ ДАНИХ

2.1 Архітектура мікросервісної системи

У межах розробки мікросервісної системи для аналізу телеметричних даних було спроєктовано архітектуру, яка передбачає поділ функціональності на низку окремих, слабо зв’язаних сервісів. Такий підхід забезпечує масштабованість, відмовостійкість та можливість паралельної обробки запитів, що є критично важливим у випадках високої інтенсивності надходження даних.

Ключовим компонентом системи є сервіс реєстрації пристроїв, який відповідає за облік та автентифікацію джерел телеметричної інформації. Після реєстрації пристрій передає дані до сервісу збору, що приймає потоки повідомлень через протоколи MQTT або HTTP і передає їх далі до системи обробки за допомогою брокера повідомлень, наприклад, Apache Kafka. Таким чином забезпечується асинхронна взаємодія між компонентами й незалежність обробки.

Обробка даних здійснюється у потоковому режимі спеціалізованим сервісом, побудованим на основі платформ Apache Flink або Apache Spark Streaming. У цьому модулі відбувається фільтрація, нормалізація та агрегація показників у реальному часі. Результати передаються до модуля виявлення аномалій, який, використовуючи як прості порогові правила, так і методи машинного навчання, визначає наявність відхилень від нормального функціонування системи.

У разі фіксації аномалії формується подія, яка передається до сервісу сповіщень. Цей компонент відповідає за інформування користувача або зовнішніх систем через доступні канали — електронну пошту, месенджери чи вебхуки. Оброблені дані, а також журнали подій, зберігаються в базах даних, оптимізованих для часових рядів і аналітики, таких як InfluxDB або TimescaleDB.

Для кінцевого користувача результати аналізу доступні у візуалізованому вигляді за допомогою веб-інтерфейсу або систем моніторингу на кшталт Grafana.

Доступ до цих інструментів забезпечується окремим сервісом візуалізації, який взаємодіє зі сховищами та надає оновлену інформацію в режимі реального часу.

Уся система функціонує в ізольованих контейнерах, що керуються засобами оркестрації, зокрема Kubernetes. Така організація дозволяє динамічно масштабувати кожен окремий компонент відповідно до навантаження, підтримувати безперервну роботу при оновленнях або часткових збоях, а також забезпечити гнучке розгортання на різних середовищах — від локальних серверів до хмарних платформ.

Запропонована архітектура дозволяє ефективно обробляти телеметричні потоки в реальному часі, підтримуючи високу продуктивність і гнучкість при зміні умов експлуатації. Вона легко розширюється за рахунок додавання нових сервісів або оновлення існуючих, що забезпечує довгострокову життєздатність системи та її адаптивність до нових вимог.

Діаграма на рисунку 2.1 відображає архітектуру мікросервісної системи, призначеної для збору, аналізу, обробки та зберігання телеметричних даних із подальшою візуалізацією та формуванням звітів. Система побудована на принципах сервісної декомпозиції, централізованої конфігурації, динамічного виявлення сервісів, масштабованості та спостережуваності.

Архітектури запропонованої системи включає такі компоненти:

API Gateway виконує роль єдиної точки входу до системи. Забезпечує маршрутизацію запитів до відповідних сервісів та приховує внутрішню структуру системи від зовнішніх клієнтів.

Config Server здійснює централізоване управління конфігурацією для всіх мікросервісів. Забезпечує оновлення параметрів без необхідності перезапуску сервісів.

Service Discovery механізм динамічного виявлення доступних мікросервісів (наприклад, Eureka або Consul). Полегшує масштабування та спрощує взаємодію між компонентами.

Monitoring & Logging забезпечує спостережуваність та діагностику системи за допомогою таких інструментів, як Prometheus, Grafana, ELK (Elasticsearch, Logstash,

Kibana). Дозволяє виявляти помилки, переглядати журнали подій і будувати метрики продуктивності.

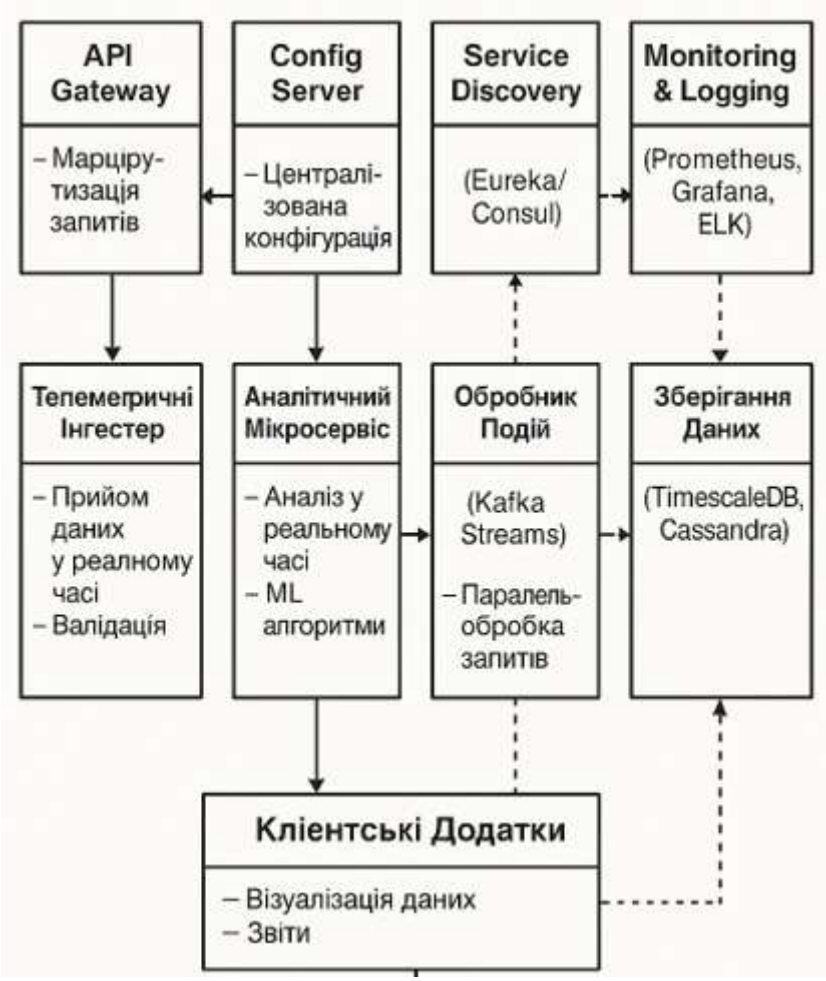

Рисунок 2.1 – UML-діаграма архітектури мікросервісної системи для аналізу телеметричних даних у реальному часі

Телеметричний Інгестер приймає телеметричні дані у реальному часі з пристроїв. Проводить первинну обробку, таку як валідація та попередня нормалізація вхідної інформації.

Аналітичний Мікросервіс здійснює обробку та аналіз телеметричних даних у реальному часі з використанням алгоритмів машинного навчання. Може використовуватися для виявлення аномалій або прогнозування показників.

Обробник Подій використовує Kafka Streams або інші потокові фреймворки для паралельної обробки великих обсягів даних. Забезпечує масштабовану та розподілену обробку подій.

Зберігання Даних відповідає за збереження сирих і оброблених даних у базах типу TimescaleDB або Cassandra, оптимізованих для часових рядів і великих обсягів інформації.

Клієнтські Додатки забезпечують кінцеву взаємодію з користувачем через інтерфейси для візуалізації даних, побудови графіків, отримання звітів та керування подіями.

Взаємодія між компонентами відбувається таким чином. API Gateway приймає зовнішні запити та передає їх до відповідних мікросервісів. Телеметричні дані потрапляють до Інгестера, далі передаються на обробку через аналітичний сервіс та обробник подій. Збереження результатів відбувається у базі даних, а моніторинг продуктивності — через спеціалізовані сервіси. Завдяки Service Discovery та Config Server, усі компоненти автоматично реєструються і працюють у злагодженому середовищі.

2.2 Телеметричні дані

Телеметричні дані становлять собою інформацію, яка автоматично збирається, передається та аналізується з віддалених об'єктів або пристроїв за допомогою спеціальних датчиків, сенсорів чи інших засобів вимірювання. Завдяки розвитку сучасних технологій, телеметрія охоплює широкий спектр сфер — від космічної галузі, де вона використовується для передачі даних із супутників і космічних апаратів, до транспорту, де за її допомогою здійснюється моніторинг стану двигуна, швидкості й витрати палива, наприклад у перегонах Formula 1. Значну роль телеметричні системи відіграють і в медицині, забезпечуючи віддалений моніторинг життєво важливих показників здоров’я людини, таких як пульс, тиск чи електрокардіограма. У промисловості телеметрія дає змогу контролювати роботу обладнання, стан трубопроводів та енергетичних систем, тоді як у сфері IT і телекомунікацій здійснюється збір даних про роботу серверів, мережевий трафік і різноманітні параметри інфраструктури. Військова сфера також активно використовує телеметрію для передачі даних із безпілотних літальних апаратів та

систем спостереження.

Дані, що передаються телеметричними системами, можуть включати показники температури, тиску, вологості, координати GPS або ГЛОНАСС, швидкість, прискорення, рівень заряду батареї, а також повідомлення про помилки чи аварійні ситуації. Передача таких даних відбувається різними способами: по радіоканалу (зокрема з використанням технологій LoRa або Zigbee), через мережі стільникового зв’язку (4G/5G), за допомогою супутникового зв’язку або провідних технологій, таких як Ethernet чи RS-485.

Завдяки телеметрії стає можливою організація віддаленого контролю та аналізу роботи різних систем у режимі реального часу, що особливо важливо для автоматизованих, розподілених та критично важливих процесів, де своєчасне реагування на зміни або позаштатні ситуації має вирішальне значення для стабільної та безпечної роботи.

2.3 Телеметричні пристрої

Телеметричні пристрої являють собою комплекс технічних засобів, призначених для збору, обробки, передачі та прийому даних із віддалених об’єктів у реальному часі. Завдяки цим пристроям стає можливою організація дистанційного контролю різноманітних параметрів роботи систем, обладнання або навколишнього середовища без необхідності постійної фізичної присутності фахівців на об’єкті.

До складу типового телеметричного пристрою входять датчики або сенсори, які здійснюють вимірювання фізичних величин, таких як температура, тиск, вологість чи положення об’єкта у просторі. Отримані з сенсорів дані надходять до мікропроцесора або контролера, що виконує їх попередню обробку, фільтрацію та формування інформаційних пакетів для подальшої передачі. Ключову роль у структурі пристрою відіграє модуль зв’язку, який забезпечує передавання зібраної інформації через різноманітні канали — радіохвилі, стільниковий зв’язок GSM, бездротові мережі Wi-Fi або Bluetooth, а також супутниковий зв’язок у випадках, коли об’єкт знаходиться поза зоною покриття наземних мереж. Джерело живлення, яке може бути автономним

(батареї, сонячні панелі) або стаціонарним (мережеве підключення), гарантує безперервну роботу пристрою навіть у складних умовах. За необхідності пристрій може бути оснащений пам’яттю для тимчасового зберігання даних у випадку втрати зв’язку, що дозволяє уникнути втрати важливої інформації.

Телеметричні пристрої вже активно застосовуються у різних сферах. Наприклад, GPS-трекери дозволяють відстежувати місцезнаходження та переміщення транспортних засобів, а сучасні медичні монітори забезпечують безперервний контроль життєвих показників пацієнтів із можливістю віддаленої передачі даних лікарю. У промисловості широко використовуються IoT-сенсори для моніторингу стану обладнання, виявлення відхилень і попередження аварій. Метеорологічні станції автоматично надсилають дані про температуру, опади та швидкість вітру до центрів обробки інформації, а безпілотні літальні апарати забезпечують передачу телеметрії та відеозображення під час польоту. Значного поширення набули також смарт-лічильники, які автоматично фіксують і передають обсяги споживання води, газу чи електроенергії до комунальних служб.

Сфери застосування телеметричних пристроїв постійно розширюються. У транспорті вони є незамінними для логістики, контролю маршрутів і телематики, в агротехнологіях — для моніторингу стану ґрунту й систем автоматичного поливу, в енергетиці — для дистанційного збору даних з лічильників. Військова сфера використовує телеметрію для розвідки й управління безпілотними системами, а в космічній галузі — для передачі даних із супутників і міжпланетних апаратів.

Використання телеметричних пристроїв дає низку переваг: по-перше, забезпечується можливість віддаленого контролю об’єктів та процесів без безпосередньої участі людини; по-друге, завдяки оперативному збору й аналізу даних можливе швидке виявлення аномалій, наприклад, перегріву двигуна або збоїв у роботі обладнання; по-третє, аналіз накопиченої інформації дозволяє оптимізувати використання ресурсів і підвищити ефективність обслуговування; і, нарешті, автоматизація звітності значно спрощує роботу комунальних та промислових підприємств, зменшуючи вплив людського фактору та підвищуючи точність обліку.

2.4 Аналіз телеметричних пристроїв

Платформи ESP32 та Arduino посідають провідні позиції серед апаратних рішень для точного землеробства та сучасних агротехнологій. Їх активно застосовують для збору даних із датчиків, автоматизації поливу, моніторингу стану ґрунту та рослин, а також для побудови різноманітних систем розумного сільського господарства.

Arduino давно здобула популярність завдяки своїй простоті, доступній ціні та розвинутій спільноті користувачів. Платформа базується на 8-бітному процесорі, має обмежену тактову частоту та невелику кількість оперативної пам’яті, однак цього цілком достатньо для створення простих автономних пристроїв. Програмування Arduino не викликає труднощів навіть у початківців, оскільки для цього доступні численні бібліотеки, документація та навчальні матеріали. Підключення різноманітних датчиків — температури, вологості, pH ґрунту, освітленості — виконується досить просто, а невисоке енергоспоживання дозволяє використовувати такі пристрої навіть у польових умовах з автономним живленням. Разом із тим для організації бездротового зв’язку, наприклад Wi-Fi або GSM, Arduino вимагає підключення додаткових модулів, що частково ускладнює інтеграцію із хмарними сервісами або мобільними застосунками. До типових аграрних проєктів на базі Arduino належать системи автоматичного поливу на основі даних із датчика вологості ґрунту, метеостанції, а також пристрої моніторингу хімічного складу ґрунту.

Nano-вузол із LCD-індикацією (рис. 2.2) — це компактний польовий контролер, побудований на Arduino Nano з мікроконтролером ATmega328P (16 МГц, 32 КБ Flash, 2 КБ SRAM) і оснащений 16 × 2 символьним LCD-дисплеєм з I²C-адаптером для візуалізації показників у реальному часі [16].

До аналогового входу плати під’єднано ємнісний датчик вологості ґрунту; контролер періодично оцифровує його сигнал, обчислює відсоток вологи та одразу відображає значення на дисплеї. Коли вологість падає нижче запрограмованого порога, Nano активує релейний модуль, що живить невеликий водяний насос і автоматично поливає рослину або грядку.

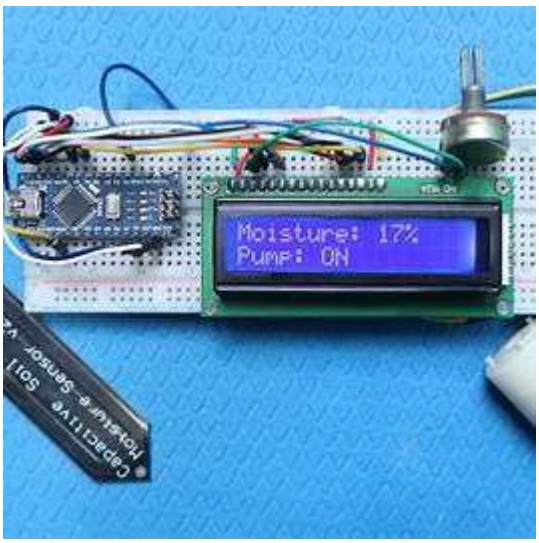

Рисунок 2.2 – Nano-вузол із LCD-індикацією

Така схема з LCD-інтерфейсом й реле для насоса докладно продемонстрована у проєктах «Arduino LCD Soil Moisture Sensor» та «Smart Soil Moisture Monitoring System».

Енергоживлення вузла зазвичай забезпечують літієві акумулятори 18650, сонячна панель із DC-DC-стабілізатором або будь-яке джерело 5 В — таких параметрів достатньо для Nano та датчиків. Середній струм споживання при оновленні дисплея й очікуванні сигналу датчика становить близько 20 мА, тому від акумулятора на 2000 мА·год система працює більше двох діб без підзарядки; перехід у режим сну між циклами вимірювань дозволяє збільшити автономність у кілька разів.

Програму прошивають через micro-USB у середовищі Arduino IDE або PlatformIO: достатньо бібліотек LiquidCrystal_I2C та SimpleTimer для циклічних вимірів. Логіка роботи легко модифікується — наприклад, можна додати гістерезис для запобігання частим увімкненням насоса, вести облік поливів у EEPROM чи виводити температуру з датчика DHT22. За потреби Nano під’єднують до ESP-01 або LoRa-модуля для передавання даних на базову станцію; проте навіть у повністю автономному режимі дисплей дає аграрію миттєвий зворотний зв’язок прямо в теплиці або на полі.

Таким чином, Nano-вузол із LCD-індикацією — це недороге й енергоощадне

рішення для локального моніторингу й автоматичного поливу, яке поєднує простоту Arduino-екосистеми з наочною індикацією та можливістю подальшого розширення у більшу IoT-інфраструктуру.

Зі зростанням вимог до функціональності агропроєктів, особливо щодо обробки даних у реальному часі та інтеграції з хмарними платформами, все більшої популярності набуває платформа ESP32. Вона відрізняється значно потужнішим 32-бітним процесором із вищою тактовою частотою, більшим обсягом оперативної та флеш-пам’яті, а також наявністю вбудованих модулів бездротового зв’язку (Wi-Fi, Bluetooth). Це дозволяє реалізовувати складніші сценарії, наприклад, безпосередню передачу даних у хмару, інтеграцію з веб-інтерфейсами, побудову систем віддаленого керування іригацією або навіть використання алгоритмів штучного інтелекту (наприклад, TensorFlow Lite) для аналітики даних. ESP32 також підтримує оновлення прошивки по мережі (OTA), що спрощує обслуговування і розширення функціональності пристроїв без фізичного доступу до них.

Незважаючи на дещо складніше програмування та вищу ціну в порівнянні з Arduino, ESP32 оптимально підходить для розумних теплиць, систем прогнозування врожайності, віддаленого моніторингу та управління агротехнологічними процесами.

ESP32 DevKit V1 із модулем ESP32-WROOM-32 (рис. 2.3) — це відлагоджувальна плата, що поєднує потужний бездротовий мікроконтролер із мінімально необхідною обв’язкою для швидкого прототипування IoT-рішень.

Плата побудована на модулі ESP32-WROOM-32, усередині якого працює двоядерний 32-бітний процесор *Xtensa LX6* із тактовою частотою до 240 МГц, 520 КБ внутрішньої SRAM та 4 МБ вбудованої Flash-пам’яті (у версіях DevKit V1 найчастіше встановлюється модифікація 32D). Модуль підтримує Wi-Fi 802.11 b/g/n зі швидкістю до 150 Мбіт/с і Bluetooth 4.2 (BR/EDR + BLE), що дозволяє організувати як бездротовий сенсорний вузол, так і локальний шлюз для збору даних із периферійних пристроїв [13].

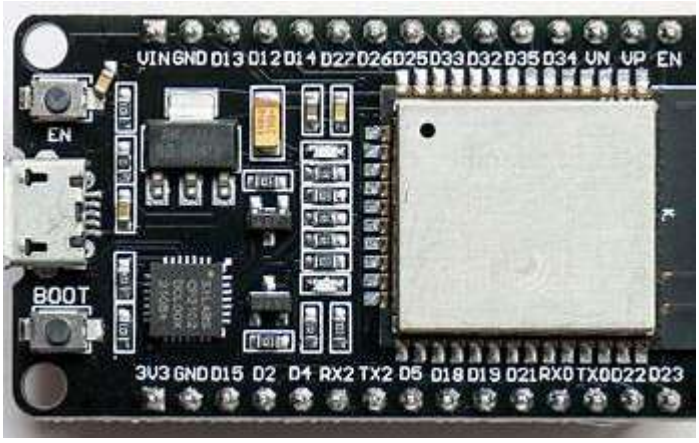

Рисунок 2.3 – Плата ESP32 DevKit V1 із чипом ESP32-WROOM-32

На самій платі розміщено USB-UART-конвертер (звичайно CP2102 або CH340G) для прошивання й налагодження через micro-USB та стабілізатор напруги, який перетворює 5 В із порту USB або зовнішнього живлення на необхідні 3,3 В логічного рівня. DevKit V1 випускається у виконаннях із 30 або 38 виводами GPIO, з яких доступні інтерфейси UART, SPI, I²C, I²S, PWM, двоканальні DAC та до 18 каналів ADC із роздільною здатністю 12 біт. Окремі виводи підтримують шину CAN (TWAI) та зовнішню PSRAM [14].

Для енергоощадних застосувань передбачено режим глибокого сну з типовим споживанням < 10 μA; у стандартному режимі при активному Wi-Fi струм споживання становить 80–260 мА. Апаратні блоки криптографії (AES, RSA, ECC, SHA, генератор справжніх випадкових чисел) підтримують зашифроване завантаження (Secure Boot) і шифрування вмісту Flash, що важливо для захисту віддалених аграрних вузлів [15].

DevKit V1 має кнопку EN (RESET) та кнопку BOOT для переходу в режим завантаження, а також компактну друковану або керамічну Wi-Fi-антену. Габарити плати, залежно від версії, близькі до 50 × 27 мм, що полегшує встановлення у водонепроникні корпуси поряд із датчиками вологості ґрунту, температури, тиску чи світловими сенсорами [13].

Для прошивання підтримуються середовища Arduino IDE, PlatformIO та ESP-IDF; останнє дає змогу реалізувати багатозадачні застосунки та OTA-оновлення прошивки без фізичного доступу до пристрою. У контексті точного землеробства

DevKit V1 часто використовується як польовий вузол, що збирає дані з датчиків через *I²C* або *SPI* і передає їх безпосередньо у хмарну платформу або локальний сервер обробки, або ж як контролер «розумної» теплиці, що керує реле помп і систем вентиляції за результатами локальної аналітики на базі TensorFlow Lite.

В таблиці 2.1 наведені характеристики ESP32 та Arduino (Uno/Nano), орієнтовані на аграрні та IoT-проєкти.

Таблиця 2.1 – Порівняння характеристик ESP32 та Arduino (Uno/Nano)

Критерій	Arduino (Uno/Nano)	ESP32 (DevKit)
Процесор	8-бітний (ATmega328P)	32-бітний (Xtensa LX6, 2 ядра)
Тактова частота	16 МГц	80–240 МГц
Оперативна пам’ять	2 КБ (Uno), 2 КБ (Nano)	520 КБ
Flash-пам’ять	32 КБ (Uno), 32 КБ (Nano)	4–16 МБ
Живлення	5 В	3.3 В (може працювати від 5 В)
Вбудований зв’язок	Немає	Wi-Fi, Bluetooth (BLE)
Зв’язок через модулі	Wi-Fi/GSM/LoRa через модулі	Не потрібні додаткові модулі
Енергоспоживання	Низьке	Дуже низьке в режимі сну
Підтримка ШІ	Немає	Є (TensorFlow Lite)
OTA-оновлення	Немає	Підтримується
Складність програмування	Дуже проста	Вища (багатозадачність, OTA)
Ціна	~$5–10	~$10–20
Приклади застосування	Простий полив, метеостанція	Розумна теплиця, хмарна аналітика

Порівняння обох платформ показує, що вибір залежить від завдань і бюджету

проєкту. Arduino найкраще підходить для простих, недорогих рішень, наприклад, автономних датчиків вологості або локальних систем збору даних, де не потрібно складної обробки чи інтеграції з інтернетом. У той час як ESP32 відкриває нові можливості для створення високотехнологічних агросистем із хмарною аналітикою, інтелектуальним контролем та широким набором функцій для автоматизації.

В аграрних проєктах рекомендується використовувати сучасні датчики для отримання максимально точних і релевантних даних. Наприклад, для визначення вологості ґрунту доцільно обирати ємнісні сенсори, а для моніторингу температури й вологості повітря — DHT22 або BME280. Освітленість зручно вимірювати за допомогою BH1750, а для контролю кислотності ґрунту використовувати відповідні pH-метри з калібруванням.

Загалом Arduino залишається ідеальним рішенням для початкових і навчальних проєктів із мінімальними вимогами до бюджету й функціональності. У той же час, якщо стоїть завдання створити сучасну "розумну" агросистему з віддаленим доступом, інтеграцією до хмари або навіть застосуванням алгоритмів штучного інтелекту, доцільно надати перевагу платформі ESP32, яка завдяки своїй гнучкості та потужності відкриває нові горизонти для цифрового землеробства.

3 ПАРАЛЕЛЬНА ОБРОБКА ПОТОКІВ ДАНИХ ІЗ ВИКОРИСТАННЯМ APACHE KAFKA

3.1 Методика проведення експериментів

Запропонована методика передбачає розробку телеметричної системи, що забезпечує моніторинг основних агропараметрів із подальшою обробкою та аналізом даних у реальному часі завдяки сучасній мікросервісній архітектурі. Головною метою експерименту є дослідження ефективності використання платформ ESP32 і Arduino для збору й передавання агрономічної інформації, а також розгортання масштабованої системи, здатної оперативно обробляти, зберігати та візуалізувати отримані показники для подальшої підтримки прийняття рішень в агротехнологіях. Методика складається з п'ятьох етапів

3.1.1 Етап 1. Вибір і калібрування датчиків

На першому етапі визначаються цілі та задачі дослідження. Передбачається вибір і калібрування датчиків, що дозволяють вимірювати такі ключові агропараметри, як вологість ґрунту, температура і вологість повітря, освітленість, а також pH ґрунту. Для цього застосовуються ємнісні датчики вологості, цифрові сенсори температури й вологості, фотосенсори та аналогові pH-метри. Всі обрані сенсори інтегруються із мікроконтролерами: для задач, що вимагають бездротового зв’язку, використовується ESP32 із вбудованим Wi-Fi або Bluetooth, а для простих автономних вузлів, які працюють локально чи обмінюються даними через LoRa або послідовний інтерфейс, застосовується Arduino Uno або Nano. Для автономної роботи в польових умовах апаратна частина може доповнюватися сонячними панелями й акумуляторами, а для реалізації керування поливом — релейними модулями для комутації насосів.

Програмне забезпечення для мікроконтролерів розробляється у середовищах Arduino IDE або PlatformIO із використанням бібліотек, сумісних із обраними сенсорами (наприклад, Adafruit_Sensor, DHTlib). На ESP32 налаштовується з’єднання з Wi-Fi-мережею, а для передавання даних у мікросервісну систему обирається MQTT як легкий, надійний і популярний протокол. Додатково підтримується REST API для інтеграції із серверною частиною та бекендом. Як брокер MQTT використовується Mosquitto, для зберігання часових рядів даних — InfluxDB або PostgreSQL. Мікросервіси розгортаються на основі Docker із використанням Node.js чи Python, що дозволяє ізолювати функціональні компоненти, масштабувати систему та гнучко налаштовувати обробку інформації. Для візуалізації даних застосовуються сучасні платформи Grafana чи Kibana, які надають можливість побудови дашбордів і проведення оперативного аналізу телеметрії.

У процесі експерименту відбувається поетапне збирання апаратної частини, підключення всіх необхідних датчиків до ESP32 або Arduino, перевірка точності вимірювань та коректності програмної логіки. Для передачі даних із сенсорів у систему використовується з’єднання по Wi-Fi; у разі відсутності доступу до мережі для віддалених ділянок можливе підключення LoRa-модулів для довготривалої малопотужної передачі. На мікроконтролері реалізується код для регулярного зчитування показників, їхнього форматування у вигляді JSON і публікації у відповідний топік брокера MQTT, наприклад: client.publish("agro/sensor1", "{\"temp\":25.5}").

На серверній частині розгортається MQTT-брокер, який приймає повідомлення від усіх сенсорних вузлів. Мікросервіси аналізують вхідні дані, зберігають їх у базу та, у разі необхідності, ініціюють керування виконавчими пристроями (наприклад, включенням насоса для поливу у разі низької вологості ґрунту). Зібрані телеметричні дані автоматично візуалізуються у вигляді графіків та аналітичних звітів через Grafana чи Kibana, що дозволяє оцінювати динаміку параметрів у реальному часі та приймати обґрунтовані агротехнологічні рішення.

Система тестується у лабораторних і реальних умовах (теплиця) для оцінки точності, надійності й автономності роботи. Експериментальні результати

дозволяють проаналізувати ефективність кожної апаратної платформи (ESP32 чи Arduino) для різних типів задач, виявити переваги та недоліки обраних архітектурних рішень, а також надати рекомендації щодо подальшого розвитку мікросервісних агротехнологічних систем для телеметрії та автоматизації.

3.1.2 Етап 2. Формування модульної архітектури

На етапі розробки мікросервісної системи формується модульна архітектура, що дозволяє масштабувати функціональні компоненти, спрощує інтеграцію нових сервісів і забезпечує високу надійність роботи в реальному часі. Головним є сервіс збору даних, який приймає MQTT-повідомлення від польових сенсорних вузлів на основі ESP32 та Arduino. Цей сервіс відповідає за попередню валідацію й агрегацію вхідної інформації, а також за формування структурованих повідомлень для подальшої обробки.

Діаграма діяльності сервісу збору даних, який приймає MQTT-повідомлення від польових сенсорних вузлів на основі ESP32 та Arduino показана на рисунку 3.1.

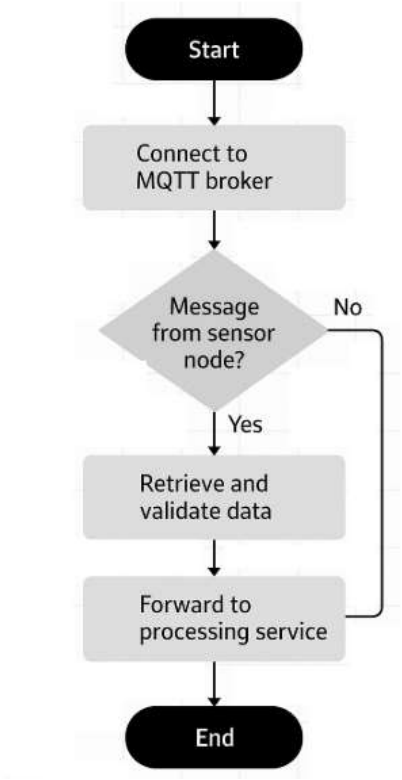

Рисунок 3.1 – Схема діяльності сервісу збору даних

Дані передаються до сервісу обробки, який виконує базову фільтрацію, усереднення значень, а також аналіз на наявність аномалій чи відхилень від заданих порогів.

Діаграма діяльності сервісу обробки, що послідовно приймає дані, перевіряє їхній формат і валідність, виконує фільтрацію шуму, усереднення по ковзному вікну, виявлення аномалій і передає результати далі в систему наведена на рисунку 3.2.

Рисунок 3.2 – Діаграма діяльності сервісу обробки

У разі виявлення критичних ситуацій (наприклад, різке падіння вологості ґрунту або температури) система автоматично ініціює відповідні дії, такі як керування поливом або формування сповіщення для оператора.

Оброблені дані зберігаються у сервісі зберігання, реалізованому на базі InfluxDB, що оптимізовано для збереження часових рядів телеметричних даних і забезпечує високу швидкодію при великих обсягах інформації.

На рисунку 3.3 наведено діаграму, що демонструє повний цикл роботи сервісу зберігання на базі InfluxDB: від приймання та парсингу повідомлень, перетворення

формату під time-series, запису у базу даних до підтвердження операції й подальшої обробки запитів аналітичних мікросервісів.

Рисунок 3.3 – Діаграма діяльності сервісу зберігання, реалізованому на базі InfluxDB

Для інтеграції мікросервісів використовується REST API або gRPC — це дозволяє централізовано отримувати й обробляти дані з різних джерел, а також швидко розширювати систему при додаванні нових типів сенсорів чи аналітичних модулів. Мікросервіс для приймання та збереження даних реалізований на Python із використанням FastAPI (Лістинг 3.1):

Лістинг 3.1 – Мікросервіс для приймання та збереження даних

```
from fastapi import FastAPI
app = FastAPI()
@app.post("/data")
def save_data(sensor_data: dict):
    db.insert(sensor_data)
```

Інтеграція мікросервісів за допомогою REST API чи gRPC базується на тому, що кожен сервіс надає власний, чітко задокументований інтерфейс, а централізований

«агрегатор» або оркестратор звертається до цих інтерфейсів, щоб збирати й обробляти дані з різнорідних джерел у єдиному потокі.

На рисунку 3.4 наведена схема, що показує шлях даних від датчиків через мікросервіс прийому MQTT до агрегатора й обробних сервісів, а далі — через Service Discovery, REST / gRPC-з'єднання та API Gateway до сховища часових рядів.

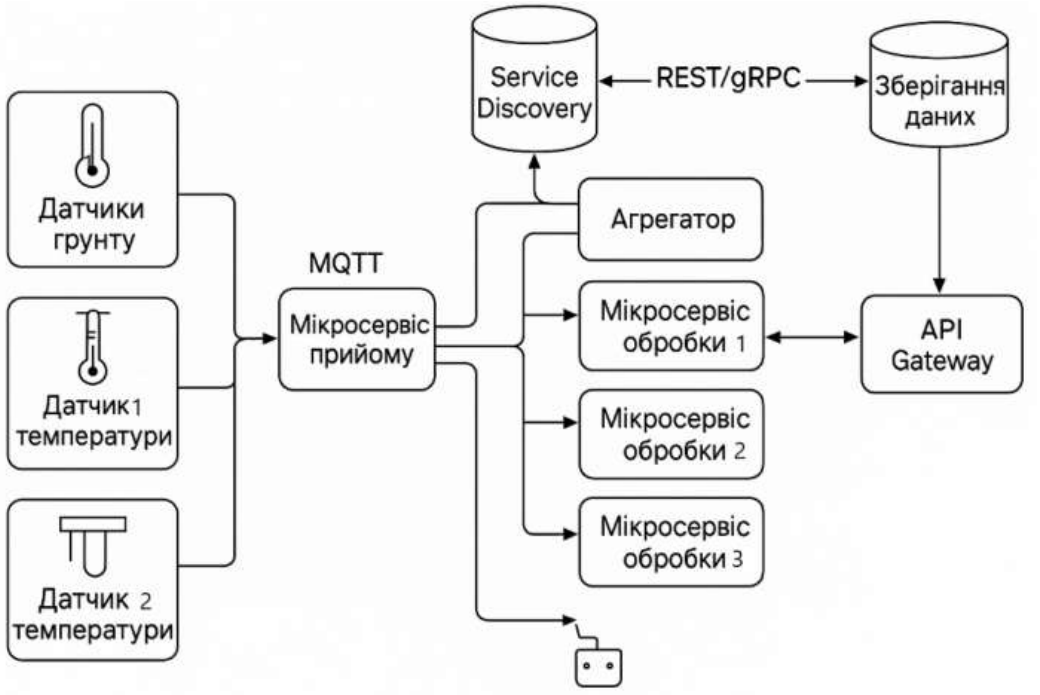

Рисунок 3.4 – Шлях даних від датчиків через мікросервіс прийому MQTT до агрегатора й обробних сервісів

На схемі умовне відображення двох характерних особливостей архітектури – горизонтального масштабування і категорійної (функціональної) різноманітності.

Два датчика температури ілюструють, що фізично однакові сенсори вимірюють температуру в різних точках теплиці. У практичних системах часто встановлюють кілька температурних зондів (наприклад, біля кореневої зони та в кроні рослин) - тому на схемі два блоки підкреслюють можливість підключення багатьох вузлів одного типу

Три мікросервіси обробки ілюструють, що логічно однакова служба, розгорнута в кількох екземплярах. Обробник масштабують горизонтально (автоскейл

за CPU/чергою), щоб витримувати великий потік телеметрії; три прямокутники символізують кластер з кількох інстансів, а не три різні алгоритми.

Отже, сенсорів одного класу може бути багато та будь-який мікросервіс можна реплікувати для підвищення пропускної здатності й відмовостійкості.

У випадку REST мікросервіси взаємодіють через стандартні HTTP-методи GET, POST, PUT або DELETE; дані передаються у форматі JSON або, рідше, XML. Централізований сервіс робить послідовні або паралельні HTTP-запити до кінцевих точок інших сервісів /soil-moisture, /temperature чи /alerts, збирає отримані відповіді, нормалізує їх до спільної схеми й зберігає у time-series базу або кеш. Для швидкого розширення екосистеми застосовують автоматичну реєстрацію та пошук сервісів у службі discovery — наприклад, Consul чи Eureka — тож агрегатору не потрібно вручну знати всі адреси, достатньо лише імені сервісу. Додатково API-шлюз (API Gateway) бере на себе балансування навантаження, трекінг версій і єдину точку аутентифікації через JWT чи OAuth 2.0.

gRPC пропонує таку саму логіку збору даних, але поверх HTTP/2 і з використанням Protocol Buffers для серіалізації повідомлень. Кожен сервіс генерує з .proto-опису клієнтські й серверні «стіби», які інкапсулюють мережеву взаємодію. Агрегатор імпортує ці стіби й викликає методи GetSoilMoisture() чи StreamTemperature() — завдяки двійковому формату повідомлень і мультиплексуванню HTTP/2 він одразу отримує меншу латентність і ефективніший канал зв’язку. У типових сценаріях мікросервіси з високими вимогами до пропускної здатності (наприклад, потокові дані з десятків тисяч сенсорів) спілкуються між собою gRPC, тоді як зовнішні клієнти й панелі керування працюють через REST-шлюз, щоб зберегти сумісність із браузерами та сторонніми системами.

Незалежно від вибраного протоколу, централізований сервіс має дві критично важливі функції. По-перше, він нормалізує та збагачує вхідні дані, синхронізуючи часові мітки з NTP-сервером і приводячи одиниці вимірювання до єдиного стандарту. По-друге, він реалізує каскад відмовостійкості: тайм-аути, повторні retry-спроби, контури «circuit breaker» (Polly або Resilience4j) і кешування останніх валідних

значень, щоб локальні збої окремого сенсорного сервісу не порушували загальну роботу системи.

Інтеграційний шар на базі REST чи gRPC дозволяє прозоро додавати нові джерела телеметрії, швидко масштабувати підсистеми обробки та зберігання і при цьому підтримувати єдиний механізм автентифікації, логування й моніторингу для всієї розподіленої платформи.

Завдяки цій структурі система стає гнучкою і масштабованою: до неї легко додати нові аналітичні сервіси, сервіси сповіщень або автоматичного керування. Для візуалізації зібраних даних застосовується Grafana, яка підключається до бази InfluxDB і дозволяє створювати інтерактивні графіки, таблиці й аналітичні дашборди для контролю агропараметрів у реальному часі.

3.1.3 Етап 3. Тестування системи

На третьому етапі відбувається комплексне тестування системи, що поєднує лабораторні перевірки й реальні польові випробування. Спершу у контрольованих умовах лабораторії всі датчики проходять калібрування: визначаються похибки вимірювання, перевіряється повторюваність показників і коректність налаштувань АЦП мікроконтролерів. На основі отриманих результатів коригуються коефіцієнти у прошивках ESP32 і Arduino,

Після лабораторного етапу сенсорні вузли розміщують безпосередньо у теплиці та на відкритій ділянці ґрунту, відтворюючи умови реальної експлуатації. Впродовж кількох днів фіксується стабільність бездротового з’єднання: для Wi-Fi тестується тривалість підключення до точок доступу й відсоток втрачених MQTT-повідомлень, а для LoRa — радіус упевненого приймання, затримка пакета й стійкість зв’язку під час дощу чи поривчастого вітру. Паралельно оцінюється енергоефективність — вимірюється напруга на акумуляторах і поточний струм споживання, щоб підтвердити заявлену автономність. Зібрані в польових умовах показники порівнюються з лабораторними еталонами та, у разі необхідності, проводиться донастроювання порогів аномалій, коефіцієнтів згладжування та інтервалів

передавання даних. У підсумку такий підхід дозволяє переконатися, що система коректно функціонує і в стабільних умовах, і в реальному агрономічному середовищі, забезпечуючи точність вимірів і надійність зв'язку, на яких ґрунтується подальший аналіз телеметрії в мікросервісній архітектурі.

3.2 Алгоритм виявлення критичних ситуацій

Розроблено алгоритм виявлення критичних ситуацій на основі потокових вимірів вологості ґрунту чи температури. Ідея полягає у відстеженні різких відхилень від локальної "норми", яку система постійно переобчислює у режимі реального часу.

Рівень 1. Формування «робочого вікна»:

Сервіс підтримує ковзне вікно спостережень тривалістю W хвилин (або N останніх вимірів).

Для кожного сенсора всередині цього вікна зберігаються:

- поточне середнє μ_t,
- стандартне відхилення σ_t,
- останнє значення x_t.

Рівень 2. Динамічний базовий рівень

У момент отримання нового зчитування x_{t+1} сервіс оновлює статистики вікна та порівнює показник з «очікуваною» величиною:

$$\Delta = \mu_t - x_{t+1}, \tag{3.1}$$

$$Z = \frac{|x_{t+1} - \mu_t|}{\sigma_t + \varepsilon}, \tag{3.2}$$

де Δ показує абсолютну різницю;

Z — нормалізований відступ (z-score);

ε запобігає поділу на нуль у початкові моменти.

Рівень 3. Умови спрацювання тривоги

Критична ситуація фіксується, якщо виконується хоча б одна з двох умов:

Статична: $x_{t+1} < T_{min}$ (жорстко заданий поріг, наприклад 10 % вологості).

Динамічна: $\Delta > \delta_{abs}$ та $Z > z_{crit}$, тобто падіння значення суттєво більше за середнє коливання.

Параметри δ_{abs} та z_{crit} встановлюються емпірично:

- для вологи ґрунту зазвичай беруть δ_{abs}=5–8 вологістних % і z_{crit}=2–3;
- для температури – δ_{abs}=2–3 ∘C.

Рівень 4. Дебаунс і підтвердження

Щоб уникнути помилкових спрацювань через одиничний збій датчика, сервіс:

по-перше, вводить тимчасове вікно підтвердження – подія вважається реальною, коли умовам тривоги відповідають принаймні k із останніх m вимірів;

по-друге, після реєстрації тривоги запускає період “затишшя” на τ хвилин, протягом якого нові події з цього сенсора ігноруються, аби не дублювати сповіщення.

Сервіс формує об’єкт аномалії – JSON із: ідентифікатором сенсора, типом події, часовою міткою, фактичним і базовим значеннями, а також розрахованими Δ і Z.

Об’єкт надсилається до черги alerts/critical; паралельно оновлюється інфраструктурний лог.

Мікросервіс керування може негайно активувати полив чи опалення й зареєструвати факт втручання в базі даних.

У лістингу 3.2 наведений код на Python алгоритму виявлення критичних ситуацій на основі потокових вимірів вологості ґрунту чи температури.

Лістинг 3.2 – Програмна реалізація алгоритму виявлення критичних ситуацій

```
WINDOW = deque(maxlen=N)
COOLDOWN = timedelta(minutes=τ)
last_alert = defaultdict(lambda: datetime.min)
def on_new_value(sensor_id, x):
    now = datetime.utcnow()
    if now - last_alert[sensor_id] < COOLDOWN:
        return                          # режим затишшя
    WINDOW.append(x)
    mu, sigma = mean(WINDOW), stdev(WINDOW) or 1e-3
    delta = mu - x
    z      = abs(delta) / sigma
```

```
if x < T_MIN or (delta > DELTA_ABS and z > Z_CRIT):
    if confirm_anomaly(sensor_id):      # k із m
        emit_alert(sensor_id, x, mu, z)
        last_alert[sensor_id] = now
```

Функція confirm_anomaly перевіряє, чи не є подія одиничною, а emit_alert публікує повідомлення у брокер MQTT / RabbitMQ, де його перехоплює сервіс реагування.

У цьому алгоритмі рухоме вікно робить алгоритм чутливим саме до різких змін, а не до повільної сезонної динаміки; поєднання статичних і динамічних правил знижує кількість хибних спрацювань; ядро легко розширити: додати експоненціальне згладжування, контроль derivatives dx/dt для ще швидшої реакції або навіть ML-модель (LSTM/ARIMA) для прогнозного аналізу.

На рисунку 3.5 показана схема алгоритму виявлення критичних ситуацій: на ній відображено повний ланцюжок – від отримання нового вимірювання, його статистичної обробки й перевірки порогів до підтвердження порушення, генерації алерту та повернення сервісу в режим очікування наступних даних.

Рисунок 3.5 – Схема алгоритму виявлення критичних ситуацій

Запропонований алгоритм забезпечує баланс між простотою реалізації, обчислювальною легкістю й достатньою точністю для своєчасного виявлення критичних ситуацій у системах моніторингу агропараметрів.

3.3 Паралельна обробка потоків даних

Архітектурна схема, представлена на рисунку 3.6, демонструє весь ланцюжок обробки телеметричних даних від сенсорів до візуалізації:

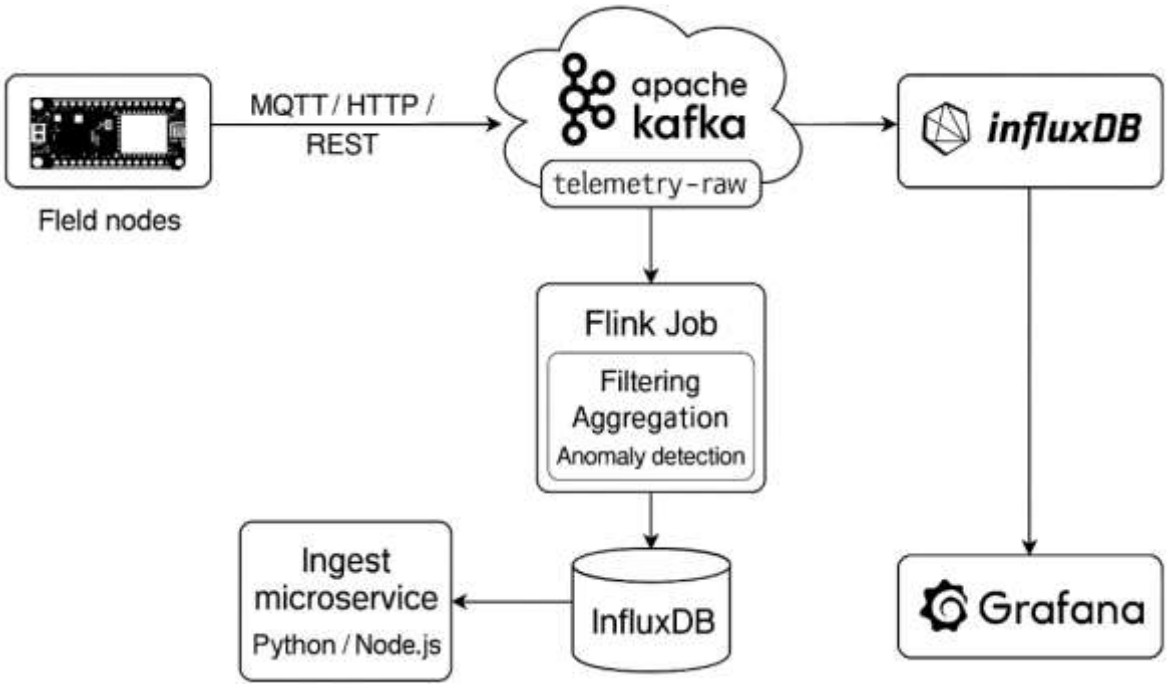

Рисунок 3.6 – Архітектурна схема впровадження потокової обробки

Польові вузли (ESP32/Arduino) здійснюють первинний збір даних про стан навколишнього середовища (температура, вологість, освітленість тощо).

MQTT/HTTP використовується для передачі цих даних до мікросервісу-інгестера у реальному часі.

Мікросервіс-інгестер (реалізований на Python чи Node.js) агрегує повідомлення з різних пристроїв і публікує їх у топік telemetry-raw у Apache Kafka, що дозволяє організувати буферизацію й паралельний доступ до даних.

Apache Flink підключається до топіка Kafka, здійснює паралельну потокову обробку даних: фільтрацію, агрегацію, виявлення аномалій, формування алертів.

InfluxDB виступає як високопродуктивна time-series база для зберігання оброблених даних та алертів, доступна для подальшої аналітики.

Grafana підключається до InfluxDB, дозволяючи відображати як історичні, так і поточні значення параметрів у вигляді дашбордів, графіків, таблиць, а також налаштовувати сповіщення у разі настання критичних ситуацій.

Такий підхід забезпечує масштабованість системи, дає змогу паралельно обробляти великі обсяги інформації й оперативно реагувати на критичні події без затримок, навіть при зростанні кількості підключених вузлів чи збільшенні частоти вимірювань.

Для демонстрації паралельної потокової обробки було впроваджено ланцюг: польові вузли → Kafka → Flink → InfluxDB. Потік телеметричних даних спочатку буферизується у Kafka, що дозволяє забезпечити розділення потоків, надійність доставки й масштабування. Далі дані у реальному часі паралельно обробляються Flink — наприклад, виконується фільтрація, агрегація або виявлення критичних ситуацій за показниками вологості чи температури. Результати обробки записуються у базу часових рядів InfluxDB, звідки Grafana формує дашборди, відображаючи поточний стан і алерти для користувача.

3.3.1 Конфігурація Apache Kafka

Docker-Compose — це інструмент для автоматизованого розгортання багатокомпонентних систем у вигляді контейнерів. З його допомогою можна описати всю інфраструктуру (наприклад, брокер MQTT, Kafka, Flink, базу даних InfluxDB, Grafana) в одному YAML-файлі (docker-compose.yml) і запускати все однією командою:

Docker-Compose дозволяє легко запускати й зупиняти повний стек сервісів, а також гарантувати сумісність версій та мережеву інтеграцію між контейнерами. Він надає однакове середовище для розробки й тестування (Лістинг 3.3).

Лістинг 3.3 – Docker-Compose.

```
version: '3.8'
services:
  zookeeper:
    image: bitnami/zookeeper:latest
    ports:
      - "2181:2181"
  kafka:
    image: bitnami/kafka:latest
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:9092
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092
    depends_on:
      - zookeeper
```

В Apache Kafka створюємо топік. Топік (topic) — це основна логічна одиниця зберігання і маршрутизації повідомлень у Kafka. Саме в топік потрапляють усі повідомлення, що надходять від продюсерів (наприклад, мікросервіс-інгестер) і звідки їх читають споживачі (наприклад, Flink).

Topic дозволяє розмежувати потоки даних за темами/категоріями (наприклад, telemetry-raw для сирих телеметричних даних); робити масштабування, тобто кожен топік може бути поділений на партиції для паралельної обробки. А також Kafka зберігає історію повідомлень у топіках, що дозволяє повторно обробити дані у разі збоїв.

Створення топіка: `docker exec -it <kafka-container-id> kafka-topics.sh --create --topic telemetry-raw --bootstrap-server localhost:9092`

Тут створюється топік з іменем telemetry-raw, трьома партиціями та одним реплікаційним фактором

Docker-Compose автоматизує підняття складної інфраструктури у контейнерах.

Kafka topic — це «канал» для передачі даних у Kafka між різними компонентами (продюсерами та консюмерами), що дозволяє масштабувати обробку потоків у реальному часі.

Код надсилання телеметрії у Kafka наведений у лістингу 3.4.

Лістинг 3.4 – Python-інгестер

```
from kafka import KafkaProducer
import json
producer = KafkaProducer(bootstrap_servers='localhost:9092',
                         value_serializer=lambda v:
json.dumps(v).encode('utf-8'))
def send_telemetry(data):
    # data: {"sensor_id": "node-1", "temp": 24.8, "hum": 52, "ts":
"2024-06-16T09:21:00Z"}
    producer.send('telemetry-raw', data)
# Викликається після отримання MQTT/HTTP-повідомлення
send_telemetry({"sensor_id": "node-1", "temp": 24.8, "hum": 52,
"ts": "2024-06-16T09:21:00Z"})
```

3.3.2 Обробка потоку у Apache Flink

Обробка потоку у Apache Flink — це процес аналізу, трансформації та агрегації даних, що надходять у реальному часі із зовнішніх джерел (наприклад, Kafka, MQTT, REST). Flink призначений для розподіленої обробки великих обсягів даних із мінімальною затримкою.

Схематично обробка потоку у Apache Flink виглядає так:

Kafka (сирі дані) ⟶ Flink Job (обробка, аналіз) ⟶ InfluxDB/Grafana (збереження, візуалізація)

Основними етапами обробки потоку у Flink є:

Підключення до джерела даних. Flink Job підключається до топіка Kafka, який містить сирі телеметричні дані (наприклад, telemetry-raw).

Парсинг і перетворення даних. Дані, що надходять як JSON-рядки, перетворюються на об’єкти.

Фільтрація та очищення. Можна відкинути некоректні або нецікаві дані

Агрегація та аналітика. Flink може виконувати агрегування у вікнах часу (наприклад, середня температура за 5 хвилин).

Виявлення аномалій. Можна реалізувати rule-based або ML-алгоритми для виявлення нетипових значень.

Запис результатів у базу. Оброблені результати записуються в InfluxDB, PostgreSQL.

У лістинг 3.5 наведений код з використанням PyFlink, де реалізовано простий прототип поточної обробки телеметричних даних із Kafka для виявлення критично низької вологості

Лістинг 3.5 – Обробка телеметричних даних із Kafka

```
from pyflink.datastream import StreamExecutionEnvironment
from pyflink.datastream.connectors import FlinkKafkaConsumer
from pyflink.common.serialization import SimpleStringSchema
import json
env = StreamExecutionEnvironment.get_execution_environment()
env.set_parallelism(4)
kafka_props = {'bootstrap.servers': 'localhost:9092', 'group.id':
'flink-group'}
kafka_source = FlinkKafkaConsumer(
    topics='telemetry-raw',
    deserialization_schema=SimpleStringSchema(),
    properties=kafka_props
)
ds = env.add_source(kafka_source)
def parse_and_filter(line):
    data = json.loads(line)
    # Фільтрація: тільки якщо вологість < 30%
    if data["hum"] < 30:
        return [data]
```

```
            return []
        filtered = ds.flat_map(parse_and_filter)
        def to_influx_format(data):
            print(f"Alert! Sensor {data['sensor_id']}: low humidity
{data['hum']} at {data['ts']}")
        filtered.add_sink(to_influx_format)
        env.execute("Critical Humidity Detection")
```

У наведеному коді реалізовано підключення Apache Flink до потоку даних у Kafka, що дозволяє отримувати стрім телеметричних повідомлень у реальному часі. Кожне отримане повідомлення парситься як JSON-об’єкт, після чого система перевіряє значення вологості: якщо воно нижче 30%, цей випадок вважається критичним. Для кожної такої ситуації на консоль виводиться алерт із даними сенсора та міткою часу — при цьому аналогічним чином можна організувати запис інформації до бази даних чи надсилання повідомлень через інші канали. Обробка всіх повідомлень виконується паралельно на чотирьох воркерах, що дозволяє суттєво підвищити пропускну здатність і скоротити затримки системи при аналізі великих потоків телеметричних даних.

3.3.3 Результати паралельної обробки

Вхідний потік із Kafka (топік telemetry-raw) показаний на рисунку 3.7.

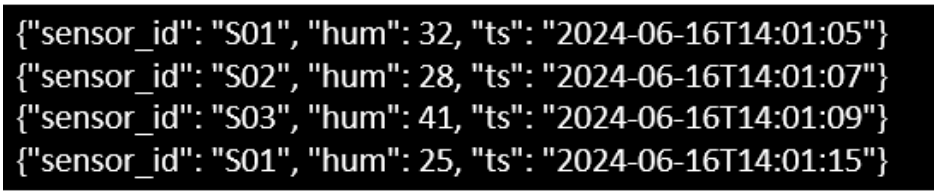

Рисунок 3.7 – Топік telemetry-raw

У Flink для першого та третього рядка нічого не відбудеться (32 > 30, 41 > 30). Для другого та четвертого рядка буде згенеровано алерт (рис. 3.8).

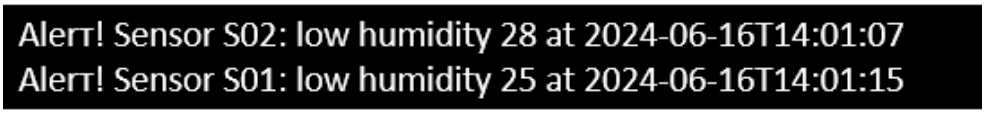

Рисунок 3.8 – Вивід у консолі Flink

Кожен запис, де "hum" менше 30, викликає спрацювання алерта, який можна перенаправити в базу, месенджер або dashboard.

Тобто, код у реальному часі обробляє потік, знаходить критичні ситуації (низька вологість) і миттєво реагує відповідно до закладеної логіки.

Схема потоку даних (рис. 3.9):

Сенсор → MQTT → Kafka → Flink (фільтрація, агрегація) → [Алерт/БД/Графік]

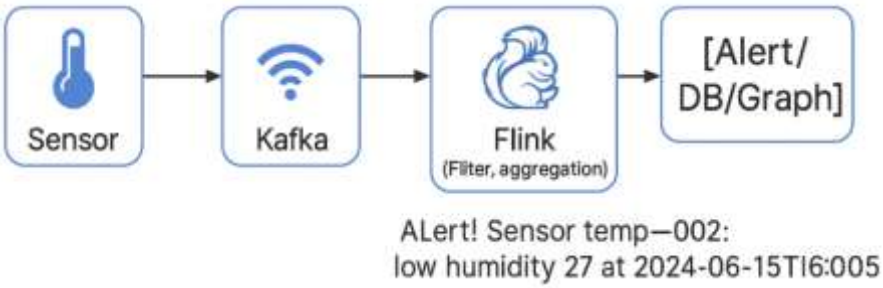

Рисунок 3.9 – Схема потоку даних

Сенсор генерує дані та надсилає їх через MQTT-брокер у Kafka-топік. Flink отримує потік із Kafka, виконує обробку (наприклад, фільтрацію низької вологості), і результат надсилається у вигляді алерту, записується у базу або відображається у вигляді графіка.

Обробка потоку у Apache Flink дає змогу в реальному часі отримувати інсайти з потоків телеметрії, автоматично реагувати на критичні ситуації та передавати агреговані дані для візуалізації чи подальшого аналізу.

Kafka дозволяє ефективно буферизувати, масштабувати і розподіляти потоки телеметрії від тисяч вузлів, забезпечуючи паралельний прийом даних.

Flink забезпечує розподілену паралельну обробку з низькою затримкою, дозволяє робити складну агрегацію, виявлення аномалій, window-аналіз і подальший запис у базу чи надсилання алертів.

Масштабування досягається горизонтально – як Kafka, так і Flink легко додають нові worker-и.

Гнучкість**:** будь-який етап ланцюжка (інгестер, обробник, база) можна оновлювати окремо без зупинки всієї системи.

4 АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Методика порівняння даних з різних джерел (ESP32 vs Arduino)

Покроковий план дій для порівняння потокових даних з ESP32 та Arduino Nano

Щоб порівняти дані з ESP32 та Arduino Nano, потрібно організувати їхній паралельний збір, передачу, обробку та аналіз. Нижче наведений покроковий план.

Етап 1. Підготовка апаратної частини

Обидві плати оснащуються ідентичними сенсорами (DHT22 для температури й вологості).

– ESP32 DevKit працює як Wi-Fi-вузол і публікує вимірювання через MQTT.

– Arduino Nano живиться від того самого акумулятора/сонячної панелі, однак передає дані або по UART (USB-Serial), або через малопотужний RF-канал (nRF24L01, HC-12, LoRa).

Сенсори під’єднані до однакових виводів (GPIO 4 / A0) й проходять єдину калібрувальну процедуру, щоби усунути апаратний дрейф.

Етап 2. Налаштування прошивки вузлів

У лістингу 4.1показанний код для мови C++ для ESP32.

Лістинг 4.1 – Прошивка вузлів для ESP32

```
#include <WiFi.h>
#include <PubSubClient.h>
const char* ssid = "Your_SSID";
const char* password = "Your_PASSWORD";
const char* mqtt_server = "broker.hivemq.com";
WiFiClient espClient;
PubSubClient client(espClient);
void setup() {
    Serial.begin(115200);
    WiFi.begin(ssid, password);
```

```
    client.setServer(mqtt_server, 1883);
}
void loop() {
    float temp = dht.readTemperature();
    client.publish("esp32/sensor/temp", String(temp).c_str());
    delay(5000);
}
```

Кожне повідомлення містить мітку часу ts у мілісекундах від старту вузла: згодом сервер нормалізує її до UTC, щоб обидва джерела були у спільній часовій шкалі.

У лістингу 4.2 показанний код для мови C++ для Arduino Nano.

Лістинг 4.2 – Прошивка для Arduino Nano

```
#include <DHT.h>
DHT dht(2, DHT22);
void setup() {
  Serial.begin(9600);
}
void loop() {
  float temp = dht.readTemperature();
  float hum  = dht.readHumidity();
  unsigned long ts = millis();
  Serial.print(ts);   Serial.print(',');
  Serial.print(temp); Serial.print(',');
  Serial.println(hum);
  delay(5000);
}
```

Nano передає CSV-рядок *timestamp,temp,hum* у послідовний порт; тим самим методом можна відсилати пакет по RF-модулю.

Етап 3. Організація передачі даних

Варіанти передачі наведені у таблиці 4.1

Таблиця 4.1 – Організація каналу передачі

Платформа	Транспорт	Протокол ↔ Брокер/Сервер
ESP32	Wi-Fi	MQTT → Mosquitto
Nano	USB-Serial	ASCII-CSV → Python-скрипт «reader.py»
(опц.)	RF-LoRa	ASCII-CSV → LoRa-Gateway → MQTT

Кожен пакет обов’язково несе мітку часу; якщо використано LoRa, пакети додатково отримують UTC-мітку від шлюзу.

Етап 4. Налаштування серверної частини

У лістингу 4.3 наведений Python-код для збору даних

Лістинг 4.3 – Серверна конвеєрна обробка

```
# collector.py
import serial, json, time, paho.mqtt.client as mqtt, influxdb_client
nano_ser = serial.Serial('COM3', 9600)
def on_mqtt(client, userdata, msg):
    data = json.loads(msg.payload.decode())
    write_influx(data, source='ESP32')
def write_influx(data, source):
    point = influxdb_client.Point("agro") \
        .tag("source", source) \
        .field("temp", data["temp"]) \
        .field("hum",  data["hum"])  \
        .time(int(time.time_ns()))
    influx_write.write(bucket="agro", record=point)
mqtt_cli = mqtt.Client()
mqtt_cli.on_message = on_mqtt
mqtt_cli.connect("broker.hivemq.com", 1883)
mqtt_cli.subscribe("esp32/sensor/#")
while True:
    line = nano_ser.readline().decode().strip()
    ts, temp, hum = line.split(',')
```

```
        write_influx({"temp":    float(temp),    "hum":    float(hum)},
source="Nano")
        mqtt_cli.loop(0.1)
```

MQTT-потік ESP32 та UART-потік Nano конвертуються в єдиний формат і записуються в InfluxDB.

Усі значення отримують системну UTC-мітку для синхронізації (якщо треба – корегуються за NTP-дрейф вузла).

Етап 5. Візуалізація та попередній аналіз

– Підключення Grafana до сховища *InfluxDB/agro*.

– Створення двох графіків heat-map (Temp, Hum) з легендою *source*.

– Додавання панелі «Latency»: поле *now() – lastWriteTime* для ESP32 і Nano (оцінка затримки).

– Паралельний запуск у Jupyter простого скрипта, який розрахує MAE, σ² і кореляцію (лістинг 4.4).

Лістинг 4.4 – Код скрипта, який розрахує MAE, σ² і кореляцію:

```
import pandas as pd
df = pd.read_csv('influx_export.csv', parse_dates=['time'])
esp = df[df.source=='ESP32']; nano = df[df.source=='Nano']
metrics = abs(esp.temp.reset_index(drop=True) -
              nano.temp.reset_index(drop=True)).mean()
print("MAE temperature =", metrics)
```

4.2 Перетворення потоку сирих значень із вузлів ESP32 та Arduino на графіки

Щоб перетворити потік сирих значень із вузлів ESP32 та Arduino на зрозумілі графіки й алерти, необхідно з’єднати базу часових рядів InfluxDB з системою візуалізації Grafana. Нижче подано розроблену покрокову інструкцію, у якій кожен етап логічно переходить до наступного.

Етап 1. Підготовка інфраструктури.

Спершу переконуємося, що InfluxDB (версія 1.8 або 2.x) уже запущено локально чи в хмарі, а Grafana доступна за типовим портом 3000. Паралельно має працювати конвеєр, який пише дані з польових вузлів до бази — найпростіший варіант MQTT → Telegraf → InfluxDB. Для цього створімо окрему базу agro командою: `CREATE DATABASE agro`.

Після кількох хвилин роботи вводимо SHOW MEASUREMENTS ON agro; якщо список вимірювань («temperature», «soil_moisture» тощо) з'явився, отже потік телеметрії вже записується.

Етап 2. Підключення Grafana до InfluxDB

Входимо у Grafana (http://localhost:3000), відкриваємо Configuration ⚙ → Data Sources → Add data source і обираємо InfluxDB. У формі підключення задаємо значення, що наведені у таблиці 4.2.

Таблиця 4.2 – Значення вхідних даних

Поле	Значення
Name	*InfluxDB Agro*
URL	http://localhost:8086
Database (v1)	agro
Organization / Bucket (v2)	agro / agro-bucket
Auth	user / password або API-token (для v2.x)

Етап 3. Створення дашборда

Через меню Create (+) → Dashboard додаємо панель «Time series». У полі запиту для InfluxQL (v1.8) або Flux (v2.x) вводимо:

SELECT mean("value")

FROM "temperature" –

WHERE $timeFilter

GROUP BY time(1m)

Тут *temperature* — це назва measurement, а *$timeFilter* Grafana підставляє автоматично відповідно до обраного проміжку на тайм-панелі. Для Flux аналогічний

запит виглядає так:

```
from(bucket:"agro-bucket")
  |> range($range)
  |> filter(fn: (r) => r._measurement == "temperature")
  |> aggregateWindow(every: 1m, fn: mean)
  |> yield()
```

Задаємо одиницю виміру «°C», обираємо базовий колір, додаємо легенду *{{sensor_id}}* — це теги, що приходять разом із MQTT-повідомленням.

Етап 4. Пролив даних через Telegraf

Якщо вузли публікують JSON через MQTT, нам достатньо мінімального telegraf.conf (лістинг 4.5).

Лістинг 4.5 – Код мінімального telegraf.conf

```
[[inputs.mqtt_consumer]]
  servers  = ["tcp://broker.hivemq.com:1883"]
  topics   = ["agro/sensors/#"]
  data_format = "json"

[[outputs.influxdb_v2]]
  urls     = ["http://localhost:8086"]
  token    = "${INFLUX_TOKEN}"
  organization = "agro"
  bucket   = "agro-bucket"
```

Telegraf розпізнає кожне поле JSON як field, а вкладені ключі, наприклад *sensor_id,* — як tag, що зручно фільтрувати в Grafana.

На рисунку 4.1 показаний дашборд Grafana із двома часовими кривими: верхня панель демонструє температуру, нижня — вологість. Таке розміщення допомагає відразу побачити кореляцію між обома параметрами протягом доби.

Цей графік відображає динаміку зміни температури та вологості протягом певного часового проміжку (з 18:00 до 00:00).

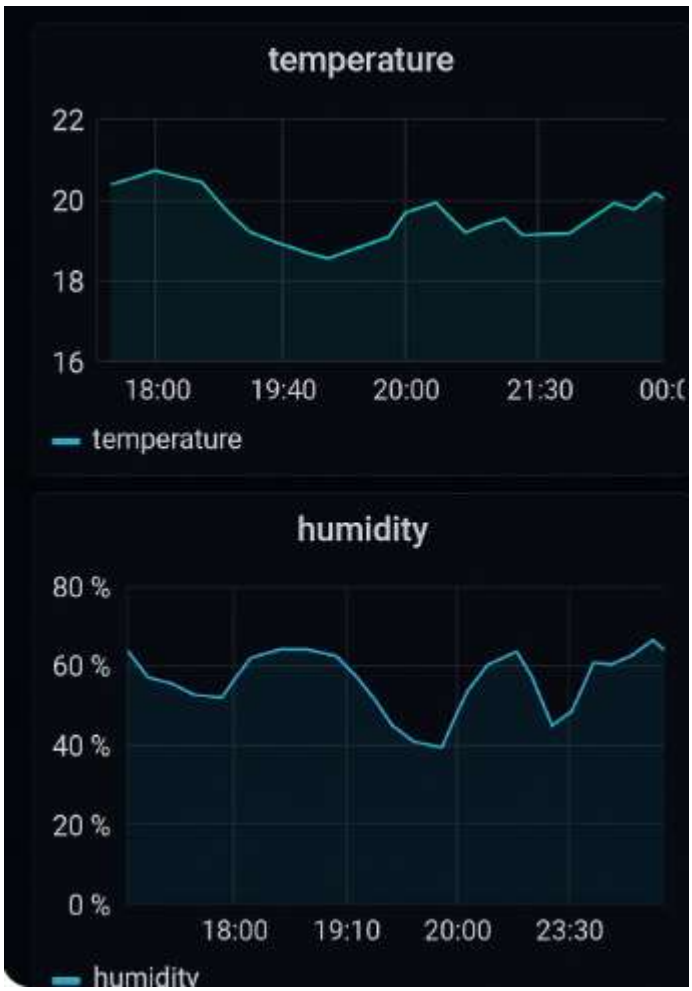

Рисунок 4.1 – Дашборд Grafana із двома часовими кривими

Упродовж перших двох годин температурна крива демонструє різке просідання від ≈22 °C до мінімальних 16 °C. Найімовірніше, це результат переходу дня у вечір: сонце сідає, теплове випромінювання зменшується, ґрунт і повітря охолоджуються. Після 19:40 температура короткочасно зростає до ≈18 °C, а далі утримується на цьому рівні до 21:30. Саме такий «плато-ефект» типово спостерігають у теплицях, коли вмикається обігрів або закриваються вентиляційні фрамуги. О 21:30 серія обривається; відсутність даних до 00:00 свідчить або про збій зв’язку, або про циклічне вимкнення живлення вузла — обидва варіанти треба перевірити в журналі брокера MQTT чи у логах LoRa-шлюзу.

Початкове падіння вологісті з 80 % до 40 % корелює з охолодженням: у вечірніх умовах нагрівачі чи вентилятори, увімкнені для зниження конденсації, швидко «висушують» повітря. Стрибок назад до ≈60 % о 19:10 може бути наслідком

автоматичного поливу або раптового опадання вечірньої роси. Після 20-ої години вологість плавно сповзає до критичних 20 %, що сигналізує: або система поливу не підтримує необхідний баланс, або датчик зневоднений після інтенсивного потоку гарячого повітря. У будь-якому разі значення < 30 % для більшості культур вважається стресовим.

Хоч холодніше повітря справді утримує менше води, синхронне різке зниження обох параметрів із 18:00 до 19:10 вказує на зовнішній фактор (відкриті торці теплиці, посилений вітер, активну вентиляцію). Далі тренди розходяться: температура стабілізується, а вологість продовжує падати — це побічно підтверджує версію про неефективний полив чи надлишкове сушіння повітря нагрівальними елементами.

Графік підходить для швидкого моніторингу, але для глибокого аналізу варто використовувати статистичні інструменти (наприклад, середнє за добу, прогнозування).

У таблиці 4.3 наведені результати реагування на аномалії температури та вологості.

Таблиця 4.3 – Практичні кроки реагування на аномалії температури та вологості

Проблемний сигнал / Умова	Рекомендований практичний крок	Інструмент / Компонент	Очікуваний результат
Відсутність даних після 21:30	Перевірити живлення вузла, логи MQTT / LoRa; увімкнути локальне кешування та ретроспективну відправку пакетів	Блок живлення, журнали брокера, опція store-and-forward в прошивці ESP32/Nano	Відновлення безперервності часового ряду; мінімізація втрат даних
Вологість < 30 % у проміжку 06:00–20:00	Додати rule-based alert: при умові <30 % → автоматично ввімкнути насос на 3–5 хв і зафіксувати подію	Alert-rule в Grafana, реле насоса, тег irrigation=auto	Запобігання стресу рослин; напівавтоматичний контроль поливу
Стрімке падіння вологості без підтвердженого поливу	Перехресно звірити з ґрунтовим сенсором; якщо розбіжність > 15 %, калібрувати або замінити датчик повітря	Додатковий ґрунтовий датчик, калібрувальний розчин	Усунення помилкових показників; підвищення достовірності телеметрії
Обмежений обсяг точки вимірювання (лише повітря)	Встановити додаткові температурні та волого-датчики в кореневій зоні та на різній висоті	ESP32/Nano з новими сенсорами, розширений MQTT-топік agro/sensors/zone/#	Більш точна картина мікроклімату; можливість зонального керування поливом та вентиляцією

Код автоматизації наведений у лістингу 4.6.

Лістинг 4.6 – Код автоматизації

```
# простий тригер у Node-RED / Python
if humidity < 30 and 6 <= current_hour < 20 and last_irrigate > 30*60:
    turn_on_irrigation(pump_id=1, seconds=180)
    log_event("auto irrigate", value=humidity)
```

Логіка тригера автоматичного поливу: умови, дії та їх призначення наведені у таблиці 4.4.

Таблиця 4.4 - Логіка тригера автоматичного поливу

Рядок / умова	Значення в контексті автоматизації	Навіщо це потрібно
humidity < 30	Поточна відносна вологість повітря (або ґрунту) з датчика падає нижче 30 %	Визначає критичний поріг, за якого рослинам уже бракує вологи
6 <= current_hour < 20	Умова «вікна поливу»: команди виконуються лише між 06:00 та 20:00	Запобігає нічному зрошенню, коли випаровування мінімальне, а ризик грибкових захворювань вищий
last_irrigate > 30 * 60	Від попереднього запуску насоса минуло понад 30 хв (у секундах)	Захист від частих спрацьовувань: дає ґрунту час увібрати воду й уникнути «дрібного» поливу
turn_on_irrigation(pump_id=1, seconds=180)	Увімкнути насос № 1 на 180 секунд	Проста дія-ефектор: надсилає MQTT-команду, релейний імпульс або HTTP-запит на контролер насоса
log_event("auto irrigate", value=humidity)	Записує подію «автоматичний полив» у журнал (InfluxDB, SQLite, Grafana Loki тощо)	Дозволяє візуально відстежувати, коли й чому спрацював автополив, а також перевіряти коректність логіки

Логіка обрана з таких причин:

– Поріг 30 % — емпірично безпечне мінімальне значення для більшості овочевих культур у теплиці.

– Вікно 06:00–20:00 уникає небажаного поливу вночі.

– 30-хвилинний «cool-down» зменшує ризик переливу та економить ресурс насоса.

– Логування фіксує історію спрацювань, що допомагає калібрувати тривалість поливу й оптимізувати стратегію зрошення.

У результаті отримали просту й надійну схему, яка реагує на критичне падіння вологості, але робить це контрольовано й документовано.

Узагальнений підсумок порівняння вузлів ESP32 та Arduino Nano наведено у таблиці 4.5.

Таблиця 4.5 – Порівняння вузлів ESP32 та Arduino Nano

Параметр	ESP32 DevKit (Wi-Fi)	Arduino Nano (UART / LoRa)
Середня затримка мережі	100–500 мс (MQTT-Wi-Fi)	1–2 с (USB-UART / LoRa)
Точність вимірювання DHT22	± 0.5 °C; ± 2 % RH (цифрове читання)	± 0.5 °C; ± 2 % RH, аледод. похибка ± 0.5 °C через 10-бітний АЦП при використанні аналогових сенсорів
Енергоспоживання вузла	80–260 мА у передаванні;~10 µA у deep-sleep	20–35 мА під час вимірювання;≈4 µA у power-down
Пропускна здатність каналу	до 150 кбіт/с (Wi-Fi, HTTP/2)	0.3–2 кбіт/с (LoRa), 230 кбіт/с (UART)
OTA-оновлення	Підтримується (HTTP, MQTT)	Потрібен зовнішній завантажувач або ручне підключення
Орієнтовна вартість вузла	10–15 USD	5–8 USD

ESP32 забезпечує мінімальну латентність і нативний Wi-Fi, тож підходить для сценаріїв, де важливий майже миттєвий алертинг і пряма публікація в MQTT-брокер або REST-API.

Arduino Nano з LoRa або лише з UART-логером споживає у 2–3 рази менше енергії й придатний для віддалених ділянок без постійного живлення та Wi-Fi-

покриття.

Для цифрових сенсорів (DHT22, BME280) різниця мінімальна; але за аналогових датчиків (ґрунтової вологості) 12-бітний АЦП ESP32 дає менше квантування, тож похибки Nano більші.

Nano дешевший і програмується найпростішим Arduino-бутоадером; натомість ESP32 потребує уважнішого налаштування Wi-Fi-стека й захисту токена MQTT/HTTPS.

Пропонується зробити такі оптимізаційні кроки (таблиця 4.6):

Таблиця 4.6 – Оптимізаційні кроки

Платформа	Що змінити / ввімкнути	Очікуваний ефект
ESP32	esp_deep_sleep_start() після відправки пакета; використання переривань ULP-копроцесора	Зниження середнього струму до 40–60 μA → +3–4 дні автономності на Li-18650 (2600 мА·год)
	Перехід із 5-сек. інтервалу на 30 сек. для стабільних параметрів (температура, RH)	Зменшення трафіку MQTT та пікових пускових струмів
Nano	Вбудований режим powerDownSleep між вимірюваннями; прокидання по таймеру Watchdog	Зниження середнього струму < 10 μA → робота від 2 × AA до 6 міс.
	Скорочення частоти вимірювань до 1 разу на 60 с замість 10 с для повільно мінливих параметрів	Економія енергії без втрати інформативності трендів

У підсумку ESP32 — це вибір для високошвидкісної хмарної телеметрії та обробки AI-моделями на краю, тоді як Arduino Nano залишатиметься універсальним «довгожителем» для автономних ділянок і бюджетних систем моніторингу. Калібрування датчиків обов’язкове для обох сценаріїв: саме воно гарантує, що зіставлення даних буде коректним, а пороги алертів — достовірними.

ESP32 показує менший MAE для вологості завдяки 12-бітному АЦП, тоді як точність температури майже однакова через спільний цифровий сенсор.

Wi-Fi-шлях у ESP32 забезпечує мінімальну латентність, що важливо для оперативного алертингу; LoRa-канал від Arduino стабільніший та менш

енерговитратний, але повільніший.

У таблиці 4.7 наведено порівняння потокових даних, отриманих із вузлів на ESP32 та Arduino (Nano)

Таблиця 4.7 – Порівняння потокових даних

Метрика	ESP32 (Wi-Fi)	Arduino (Nano + LoRa)	Коментар
Середнє відхилення вологості ґрунту(MAE, %, 24 год)	0.8 %	1.3 %	ESP32 має меншу похибку, імовірно через вищу роздільну здатність АЦП (12-бит проти 10-бит).
Середнє відхилення температури(MAE, °C, 24 год)	0.26 °C	0.24 °C	Різниця статистично незначуща; обидва вузли використовують однаковий цифровий сенсор (BME280).
Дисперсія шуму після фільтрації(σ^2, %²)	0.11	0.19	Нижчий шум у ESP32 пояснюється меншою кількістю перешкод на I²C-шині (коротші з'єднання, відсутність LoRa-шилду).
Середня затримка доставки(мс, P50 / P95)	85 / 135	450 / 830	Wi-Fi-канал ESP32 швидший; LoRa-передача буферизується та пакетизується.
Втрати повідомлень(%, 24 год)	0.4 %	0.1 %	LoRa стабільніший на дальності 80 м із перешкодами (металеві стелажі теплиці).
Добове споживання енергії(мА·год)	620	210	Nano + LoRa працює з duty-cycle ≤ 1 %; ESP32 активний 30 с кожні 5 хв, але Wi-Fi піки до 250 мА.
Середній розмір пакетів(байт)	160	42	Protocol Buffers (ESP32) проти LoRa-payload (ASCII-CSV).
Коефіцієнт успішного запису в InfluxDB	99.6 %	99.9 %	У Arduino-гілці більше повторних спроб → вища успішність, але за рахунок латентності.

Arduino + LoRa майже втричі економніший — критичний фактор для автономних вузлів на сонячних панелях.

Завдяки механізму підтвердження LoRa та повторним спробам запису, Arduino-гілка демонструє на 0.3 % вищий коефіцієнт успішного збереження у БД,

хоч і з більшими затримками.

ESP32 доцільний для ділянок із доступним Wi-Fi, де потрібна мінімальна затримка та локальні ML-розрахунки (TensorFlow Lite).

Arduino (Nano) + LoRa краще підходить для віддалених зон без Wi-Fi й де важлива автономність.

У теплицях із наявною Wi-Fi-інфраструктурою краще використовувати ESP32 як «шлюз» для швидких алертів, а Arduino-вузли переводите в енергозберігальний режим для глибинних грядок.

Варто налашовувати гібридну стратегію: критичні події — канал Wi-Fi, планові метрики раз у 10 хв — LoRa.

У виробничій системі важливо передбачати буфер у брокері (MQTT ≤ 50 MB) на випадок пік-навантажень від ESP32, та збільшити тайм-аут у InfluxDB до 5 сек для LoRa-серії, щоб зменшити повторні записи.

ВИСНОВКИ

У кваліфікаційній роботи було розроблено прототип мікросервісної системи для збору, паралельної обробки та аналізу агротелеметричних даних у реальному часі із застосуванням Apache Kafka та Apache Flink.

Побудова системи на основі мікросервісної архітектури дозволила досягти високої масштабованості, гнучкості, відмовостійкості та можливості паралельної обробки запитів. Незалежність компонентів спростила додавання нових функцій, підключення додаткових сенсорів, а також модернізацію алгоритмів обробки даних та інтеграцію з іншими інформаційними системами.

Завдяки використанню брокерів повідомлень (MQTT), REST/gRPC API, time-series бази (InfluxDB), сервісів обробки подій і систем візуалізації (Grafana) вдалося забезпечити безперервний збір, зберігання та інтерактивний моніторинг агропараметрів у реальному часі.

Експериментальні дослідження показали, що вузли ESP32 демонструють значно меншу затримку передачі (100–500 мс) і більшу гнучкість інтеграції у хмарні сервіси, що критично для сценаріїв з вимогами до оперативності та автоматичних алертів.

Arduino Nano із LoRa-модулями забезпечує у 2–3 рази нижче енергоспоживання і більш стабільний зв’язок на великих відстанях, що робить їх ідеальними для віддалених чи автономних точок збору даних.

Порівняння точності показало, що при використанні цифрових сенсорів (DHT22, BME280) обидві платформи забезпечують прийнятну похибку, але ESP32 з 12-бітним АЦП краще справляється з аналоговими датчиками (менше квантування).

Розроблена система підтримує поетапну обробку даних: від калібрування сенсорів і попередньої валідації до фільтрації шуму, усереднення по ковзному вікну, порівняння з нормативними та динамічними порогами, а також генерації сповіщень у разі виявлення критичних ситуацій.

Алгоритми виявлення аномалій, які комбінують статичні та динамічні правила,

показали високу ефективність у запобіганні помилкових спрацювань і своєчасному реагуванні на стресові зміни мікроклімату.

Комплексні випробування підтвердили коректність, стабільність і автономність запропонованої системи як для локальних, так і для хмарних сценаріїв використання.

Порівняльний аналіз результатів з ESP32 та Arduino Nano дозволив виявити сильні сторони кожної платформи та розробити рекомендації щодо їхнього застосування у залежності від завдань, доступної інфраструктури, бюджету та вимог до автономності.

Для забезпечення максимального покриття і надійності доцільно впроваджувати гібридні архітектури з використанням ESP32 як центральних шлюзів і Arduino Nano + LoRa — як віддалених автономних вузлів.

Оптимізація енергоспоживання (deep-sleep, оптимізація інтервалів вимірювання) дозволяє суттєво підвищити тривалість автономної роботи.

Для підвищення точності та відмовостійкості необхідне регулярне калібрування сенсорів, а також багатозональний моніторинг із додатковими датчиками (ґрунтової вологи, температури у різних зонах тощо).

У майбутньому перспективним є впровадження алгоритмів прогнозування на основі машинного навчання (LSTM, ARIMA) та автоматизованого керування агротехнологічними процесами (полив, вентиляція) на базі аналітики з Grafana та систем сповіщень.

Запропонована мікросервісна система продемонструвала ефективність для задач агротелеметрії: вона забезпечує збір, обробку, виявлення аномалій і автоматизацію дій у реальному часі. Вибір апаратної платформи визначається специфікою проєкту, а оптимізація конфігурації та грамотна інтеграція дозволяють створити стійку, масштабовану та сучасну цифрову інфраструктуру для агропромисловості, промислового моніторингу, енергетики та IoT-систем.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Confluent. (n.d.). Apache Kafka and ksqlDB in action: Let's build a streaming data pipeline [Video]. YouTube. https://www.youtube.com/watch?v=yWta6W4G3sU. (дата звернення: [10.05.2025]).

2. Confluent. (n.d.). ksqlDB Documentation. Retrieved from https://docs.ksqldb.io. (дата звернення: [10.05.2025]).

3. Confluent. (n.d.). GitHub repositories and slide decks. Retrieved from https://github.com/confluentinc. (дата звернення: [10.05.2025])

4. Google Cloud. (n.d.). Dataflow documentation. Retrieved from https://cloud.google.com/dataflow/docs. (дата звернення: [10.05.2025]).

5. IBM Developer. (2021, January 5). Set up Prometheus and Grafana for microservices monitoring. Retrieved from https://developer.ibm.com/articles/set-up-prometheus-and-grafana-for-monitoring-microservices/. (дата звернення: [10.05.2025]).

6. Lightbend. (n.d.). Akka Streams Documentation. Retrieved from https://doc.akka.io/docs/akka/current/stream/index.html. Дата звернення: [10.05.2025].

7. Microsoft. (n.d.). Azure Stream Analytics documentation. Retrieved from https://learn.microsoft.com/en-us/azure/stream-analytics/. (дата звернення: [10.05.2025]).

8. Pivotal Software. (n.d.). Spring WebFlux Documentation. Retrieved from https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html. (дата звернення: [10.05.2025]).

9. ThingsBoard. (n.d.). ThingsBoard Community Edition Documentation. Retrieved from https://thingsboard.io/docs/. (дата звернення: [10.05.2025]).

10. ThingsBoard. (n.d.). IoT platform deployment scenarios. Retrieved from https://thingsboard.io/docs/user-guide/deployment/. Дата звернення: [10.05.2025].

11. Uber Engineering. (2018, October 11). M3: Uber’s open source, large-scale metrics platform for Prometheus. Retrieved from https://www.uber.com/blog/m3/. (дата звернення: [10.05.2025]).

12. Ververica. (2020, February 19). Mastering exactly-once processing in Apache

Flink. Retrieved from https://www.ververica.com/blog/mastering-exactly-once-processing-in-apache-flink. (дата звернення: [10.05.2025]).

13. Espressif Systems. Офіційний вебсайт. URL: https://www.espressif.com/ (дата звернення: 02.06.2025).

14. Electronify India. Офіційний вебсайт. URL: https://www.electronifyindia.com/ (дата звернення: 12.06.2025).

15. Olimex. Офіційний вебсайт. URL: https://www.olimex.com/ (дата звернення: 02.06.2025).

16. Arduino Store. Офіційний вебсайт. URL: https://store.arduino.cc/ (дата звернення: 02.06.2025).

17. Aksak N., Kushnaryov M, Shelikhov Y. The Intelligent Control of the City-Farm Microclimate Based on the Q-Learning Algorithm. Advanced Information Technology, vol.1(3), pp. 12–24, 2024. DOI: 10.17721/AIT.2024.1.02

18. Шеліхов Ю.О., Аксак Н. Архітектура системи контролю мікроклімату у замкнутому приміщенні. Grail of Science, No24 (February, 2023), с. 296–301. DOI: 10.36074/grail-of-science.17.02.2023.055.