НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету (Директор ННІ)	**Завідувач кафедри**
Інформаційних технологій	Комп'ютерних систем, мереж та кібербезпеки
(назва факультету(ННІ))	(назва кафедри)
__________ Болбот І.М., д.т.н, проф.	_______________ Касаткін Д.Ю., к. пед.н., доц.
(підпис) (ПІБ, вчене звання і ступінь)	(підпис) (ПІБ, вчене звання і ступінь)
«___»____________2025 р.	«___»____________2025 р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: Дослідження комп'ютерної системи та інтелектуальних алгоритмів для систем з loT пристроями

Спеціальність 123 «Комп’ютерна інженерія»
(код і найменування)

Освітня програма Комп’ютерні системи та мережі
(назва)

Орієнтація освітньої програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

к.т.н., доцент	____________	Місюра М.Д.
(науковий ступінь та вчене звання)	(підпис)	(ПІБ)

Керівник магістерської кваліфікаційної роботи

к.т.н., доцент	____________	Місюра М.Д.
(науковий ступінь та вчене звання)	(підпис)	(ПІБ)

Виконав	________________	Донець М.В.
	(підпис)	(ПІБ)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп’ютерних систем, мереж та кібербезпеки
к.пед.н., доц. ____________ Касаткін Д.Ю.
(вчене звання і ступінь) (підпис) (ПІБ)
«__» ___________ 20__ р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Донець Максим Віталійович
(прізвище, ім’я, по батькові)

Спеціальність 123 «Комп’ютерна інженерія»
(код і найменування)

Освітня програма Комп’ютерні системи та мережі
(назва)

Орієнтація освітньої програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Тема магістерської кваліфікаційної роботи: «Дослідження комп'ютерної системи та інтелектуальних алгоритмів для систем з IoT пристроями»

затверджена наказом ректора НУБіП України від “29” жовтня 2024р. № 1941 «С»

Термін подання завершеної роботи на кафедру 14 листопада 2025 р.

Вихідні дані до магістерської кваліфікаційної роботи є вимоги до створення програмного емулятора IoT-подій, що моделює поведінку великої кількості віртуальних пристроїв, специфікації програмних модулів, параметри мережевої взаємодії, моделі машинного навчання для кластеризації та аналітики навантаження, а також формати телеметрії й структури подій, що передаються через MQTT/HTTP-транспорт.

Перелік питань, що підлягають дослідженню:

1. Дослідження архітектури програмного емулятора та принципів моделювання телеметричних подій.
2. Дослідження роботи алгоритмів інтелектуального аналізу телеметрії та їх продуктивності.
3. Аналіз інтеграції системи з телеметричними сервісами, брокерами та модулями моніторингу.

Перелік графічного матеріалу (за потреби) ______________________

Дата видачі завдання “ 29 ” жовтня 2024 р.

Керівник магістерської кваліфікаційної роботи ____________ Місюра М.Д.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання ____________ Донець М.В.
(підпис) (прізвище та ініціали)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ........5
ВСТУП........7
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ........9
1.3 Аналіз існуючих рішень........16
1.4 Функціональна структура та принципи роботи системи........20
1.5 Аналіз вимог системи........23
1.6 Постановка завдання........25
2 ПРОЄКТУВАННЯ ЕМУЛЯТОРА СЕРВЕРНОЇ ОБРОБКИ ПОДІЙ У КОМП'ЮТЕРНІЙ СИСТЕМІ З ІНТЕЛЕКТУАЛЬНИМИ АЛГОРИТМАМИ ДЛЯ IoT-ПРИСТРОЇВ........28
2.1 Принципова схема емулятора та структура каналів передачі подій........28
2.2 Електрична та монтажна схема дослідного стенду емулятора........30
2.3 Передумови створення програмного емулятора серверної логіки та вибір технологій........37
2.4 Моделювання предметної області емулятора серверної обробки подій........39
2.5 Формалізація специфікації повідомлень і тем MQTT........42
2.6 Висновки до другого розділу........45
3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЕМУЛЯТОРА........47
3.1 Технологічні засоби програмної реалізації інтелектуальних алгоритмів у Python........47
3.2 Архітектура програмного емулятора та проектування........49
3.3 Кластеризаційний аналіз апаратних профілів навантаження емулятора........53
3.4 Алгоритмізація програмних модулів емулятора........55
3.5 Висновки до третього розділу........58

4 ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ ЕМУЛЯЦІЙНОЇ СИСТЕМИ 60
4.1 План тестування програмних модулів та методика оцінювання результатів 60
4.2 Тестування функціонування системи та аналіз поведінки емулятора подій 62
4.3 Забезпечення апаратної безпеки та надійності емулятора 65
4.4 Результати тестування та склад інсталяційного пакета системи 66
4.5 Висновки до четвертого розділу 69
ВИСНОВКИ 71
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 74

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

1. AR — augmented reality, доповнена реальність
2. MR — mixed reality, змішана реальність
3. VR — virtual reality, віртуальна реальність
4. 3D — тривимірне геометричне подання об’єктів
5. FPS — frames per second, частота відтворення кадрів
6. VIO — visual-inertial odometry, візуально-інерційна одометрія
7. PnP — Perspective-n-Point, алгоритм визначення пози за ключовими точками
8. EPnP — Efficient PnP, оптимізований метод оцінювання поз
9. IMU — inertial measurement unit, інерціальна вимірювальна система
10. SLAM — simultaneous localization and mapping, одночасна локалізація і побудова карти
11. ORB — Oriented FAST and Rotated BRIEF, детектор та дескриптор ключових точок
12. AKAZE — Accelerated KAZE, алгоритм екстракції ключових точок
13. ArUco — маркерна система трекінгу для комп’ютерного зору
14. JWT — JSON Web Token, токен авторизації
15. SSO — single sign-on, єдиний вхід у систему
16. OIDC — OpenID Connect, протокол автентифікації
17. API — application programming interface, інтерфейс програмної взаємодії
18. SDK — software development kit, набір засобів розробника
19. GUI — graphical user interface, графічний інтерфейс користувача
20. UI — user interface, інтерфейс користувача
21. SQLite — реляційна вбудована система керування базами даних
22. ORM — object-relational mapping, об’єктно-реляційне відображення
23. PBR — physically based rendering, фізично коректний рендеринг
24. KPI — key performance indicators, ключові показники ефективності

25. Latency — затримка між обробкою кадру та його відтворенням
26. TrackingEngine — підсистема трекінгу AR-сцени
27. MathCore — обчислювальне ядро параметричних і аналітичних моделей
28. Renderer — модуль рендерингу тривимірних об’єктів
29. SceneView — вікно відображення AR-сцени в PyQt6
30. Telemetry — телеметричні дані системи (FPS, latency, track-loss, події)
31. Session — сесія роботи користувача в AR-додатку
32. RuleRegistry — модуль правил валідації кадрів і сцен

ВСТУП

Стрімкий розвиток інформаційних технологій і зокрема концепції інтернету речей (IoT) зумовив глибокі зміни у підходах до створення, інтеграції та експлуатації розподілених комп'ютерних систем. Сучасні IoT-рішення охоплюють величезну кількість сенсорних пристроїв, контролерів, шлюзів і хмарних аналітичних модулів, які спільно забезпечують безперервний збір і передачу телеметричних даних. Зростання обсягів цієї інформації вимагає нових підходів до її оброблення - не лише на рівні збереження та маршрутизації, а й на рівні інтелектуального аналізу з використанням алгоритмів машинного навчання. Традиційні централізовані архітектури не забезпечують потрібної швидкодії та адаптивності, що створює ризики перевантаження систем, втрати даних і зниження точності прогнозів. Тому актуальною науково-практичною задачею є створення комп'ютерної системи, здатної інтегрувати інтелектуальні алгоритми аналізу IoT-потоків даних у режимі реального часу для підвищення ефективності керування технічними об'єктами та технологічними процесами [1].

Метою магістерської роботи є розроблення та дослідження комп'ютерної системи з інтелектуальними алгоритмами аналізу й прогнозування параметрів у середовищі IoT-пристроїв, яка забезпечує адаптивну обробку даних, автоматичне виявлення аномалій і підвищення точності прогнозів за рахунок застосування моделей машинного навчання.

Для досягнення поставленої мети у роботі необхідно виконати такі **завдання:**

1. провести системний аналіз предметної області та існуючих архітектур IoT-систем і технологій обміну даними.
2. Розробити інформаційну, структурну та функціональну моделі комп'ютерної системи з урахуванням вимог до продуктивності й надійності.
3. Спроєктувати архітектуру системи збору, оброблення та інтелектуального аналізу IoT-телеметрії з використанням Python-технологій.

4. Реалізувати алгоритми машинного навчання для прогнозування параметрів середовища й виявлення відхилень у потокових даних.

5. Провести програмну реалізацію системи, тестування її працездатності, стабільності та швидкодії.

6. Оцінити ефективність запропонованих алгоритмів за критеріями точності прогнозу, обчислювальної складності та масштабованості системи.

Об'єктом дослідження є процес функціонування комп'ютерної системи збору, оброблення й інтелектуального аналізу даних у розподіленому середовищі IoT-пристроїв.

Предметом дослідження є методи, моделі та алгоритми машинного навчання для прогнозування, класифікації та адаптивного керування потоками даних у комп'ютерних IoT-системах.

Методи дослідження базуються на використанні системного аналізу, математичного моделювання, теорії інформаційних процесів і методів машинного навчання. Для практичної реалізації застосовано Python-середовище з бібліотеками NumPy, Pandas, Scikit-learn, TensorFlow, а також протоколи MQTT і HTTPs для комунікації між сенсорними пристроями, сервером аналітики й користувацьким інтерфейсом.

Наукова новизна отриманих результатів полягає у розробленні інтегрованої комп'ютерної системи, що поєднує архітектуру подієвого обміну даними з інтелектуальними алгоритмами машинного навчання для аналізу IoT-потоків у реальному часі. Запропоновано метод динамічної адаптації прогнозних моделей залежно від характеристик вхідних даних, що забезпечує підвищення точності й стабільності прогнозів у порівнянні з традиційними статичними підходами. Розроблена система є універсальною платформою для аналітики телеметричних даних та може бути використана у сферах енергоменеджменту, автоматизованого контролю мікроклімату, виробничого моніторингу й розумного середовища [2].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області та основних процесів функціонування системи

Предметна область дослідження охоплює процеси створення та функціонування комп’ютерної системи збору, оброблення й аналізу даних від інтелектуальних IoT-пристроїв, призначеної для моніторингу й прогнозування параметрів середовища в реальному часі. Такі системи належать до класу розподілених інформаційно-вимірювальних комплексів, у яких поєднуються апаратні сенсорні вузли, комунікаційна інфраструктура та програмно-аналітичні модулі з використанням методів машинного навчання. На відміну від традиційних систем збору даних, IoT-рішення забезпечують децентралізовану обробку телеметрії, гнучке масштабування, підвищену відмовостійкість і можливість інтеграції з хмарними сервісами.

Функціонування системи базується на концепції кіберфізичних мереж, де фізичні сенсорні пристрої (модулі вимірювання, актуатори, контролери) взаємодіють з цифровими аналітичними процесами. Польові вузли на базі мікроконтролерів ESP32 виконують вимірювання параметрів середовища (температура, вологість, тиск), передають дані через мережеві інтерфейси Wi-Fi із використанням протоколів MQTT/TLS, а отримані показники проходять послідовні етапи маршрутизації, буферизації, збереження, інтелектуальної обробки та візуалізації. Такий підхід формує замкнений контур керування, що забезпечує автоматизоване реагування на відхилення показників, сповіщення користувачів і формування прогнозів на основі накопичених даних.

Архітектура системи є багаторівневою та включає:

– польовий рівень, що складається із сенсорних вузлів та виконавчих елементів;

– мережевий рівень, який забезпечує зв’язок між пристроями, шлюзами та сервером;

– аналітичний рівень, де здійснюється обробка даних, прогнозування та генерація керувальних впливів;

– користувацький рівень, який надає доступ до аналітичних результатів через графічний інтерфейс або веб-панель.

Узагальнену структурно-апаратну організацію системи наведено на рис. 1.1, де показано взаємозв’язки між основними компонентами: сенсорними вузлами, комунікаційним середовищем, шлюзовим модулем, сервером аналітики та зовнішніми сервісами. Архітектура підтримує подієву модель обміну даними, що дозволяє забезпечити низьку затримку передачі інформації та високу надійність при роботі з великими обсягами телеметрії.

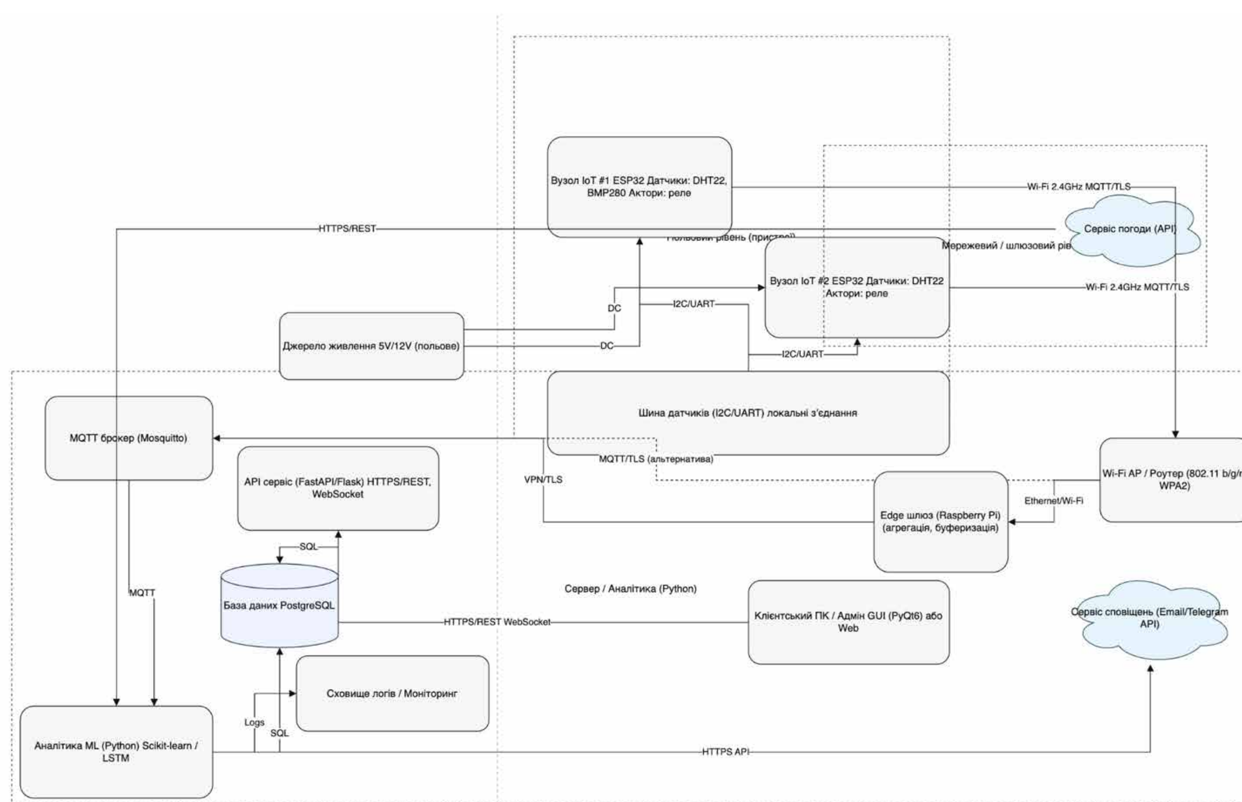


Рис. 1.1 – Апаратна схема комп’ютерної системи з інтелектуальними алгоритмами для IoT-пристроїв

Основні технічні компоненти системи подано у табл. 1.1, яка відображає призначення, тип і технічні параметри елементів, що забезпечують функціонування всієї інфраструктури.

Таблиця 1.1 – Основні апаратні складові системи

Компонент	Тип / Модель	Призначення	Основні технічні параметри
Мікроконтролер вузла IoT	ESP32-WROOM-32	Збір і передача даних до мережі	Wi-Fi 2.4 GHz, CPU 240 MHz, Flash 4 MB
Сенсор температури/вологості	DHT22	Вимірювання мікрокліматичних показників	ΔT ±0.5 °C, ΔH ±2 %
Сенсор тиску	BMP280	Вимірювання атмосферного тиску	ΔP ±1 hPa
Комунікаційний шлюз	Raspberry Pi 4 B	Агрегація, фільтрація і маршрутизація потоків	ARM CPU 1.5 GHz, RAM 4 GB
MQTT-брокер	Mosquitto	Координація потоків повідомлень між пристроями	QoS 1–2, TLS 1.3
Сервер аналітики	Python / PostgreSQL	Обробка, прогнозування, зберігання даних	ML-моделі LSTM, Random Forest
Клієнтський інтерфейс	PyQt6 / Web GUI	Візуалізація та керування системою	REST / WebSocket API

Апаратна структура забезпечує логічну взаємодію компонентів на кожному рівні системи - від сенсорного збору до аналітики. Польові вузли формують потоки телеметрії, шлюз виконує попередню агрегацію даних, сервер здійснює аналітичну обробку й прогнозування, а результати передаються користувачу через інтерактивний інтерфейс [3]. Уся комунікація відбувається в зашифрованому вигляді за допомогою протоколів TLS, що гарантує безпеку інформаційних потоків.

Предметна область дослідження визначає функціонування комплексної IoT-системи, у якій поєднуються апаратні, мережеві та програмно-аналітичні компоненти для створення автономного інтелектуального середовища моніторингу. У подальших підрозділах буде здійснено обґрунтування технічного рішення, формалізацію вимог і побудову моделей системи, які забезпечать реалізацію поставленої мети дослідження.

1.2 Теоретико-методологічні засади та стан наукових досліджень

Сучасний розвиток інтернету речей (IoT) базується на еволюції архітектурних моделей, які забезпечують узгоджену взаємодію між сенсорними вузлами, мережевими протоколами, сервісними рівнями та аналітичними модулями на основі штучного інтелекту [1–5]. Початково дослідження зосереджувалися на трирівневих структурах, що визначали послідовність оброблення даних - від фізичного шару до прикладних сервісів.

Науковці Atzori L., Iera A., Morabito G. (2010) запропонували сервісно-орієнтовану архітектуру (SOA), у якій визначено чотири базові шари - об’єкти**,** абстракцію об’єктів**,** керування сервісами та композицію сервісів. Ця модель стала теоретичним підґрунтям для побудови інтелектуальних комп’ютерних систем, орієнтованих на масштабованість, повторне використання функцій і централізоване керування ресурсами (рис. 1.2).

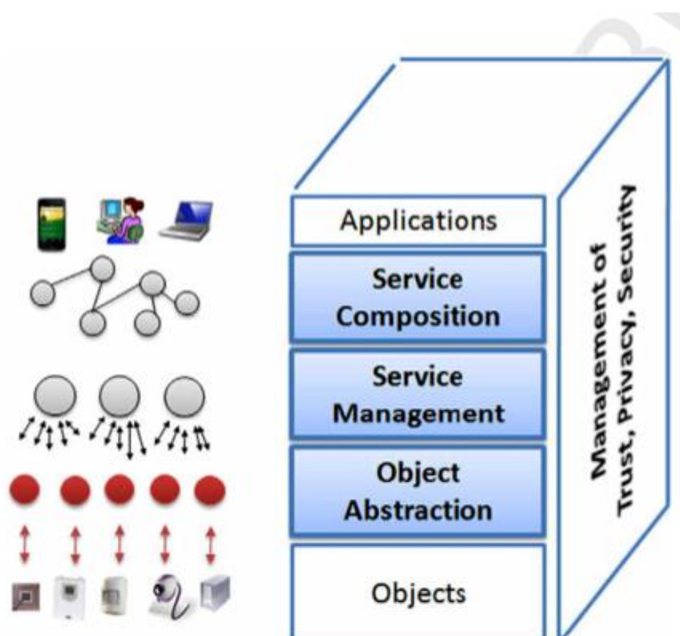


Рис. 1.2 – Сервісно-орієнтована модель архітектури інтернету речей (адаптовано за Atzori et al., 2010 [1]).

Подальші дослідження Al-Fuqaha et al. (2015) деталізували моделі IoT-систем, запропонувавши кілька архітектур: тришарову, middleware-орієнтовану, сервісно-орієнтовану та п’ятишарову. У кожній із них підкреслено різну глибину інтеграції мережевих, координаційних і прикладних рівнів, що забезпечує

підвищення ефективності інформаційних потоків у розподілених системах (рис. 1.3).

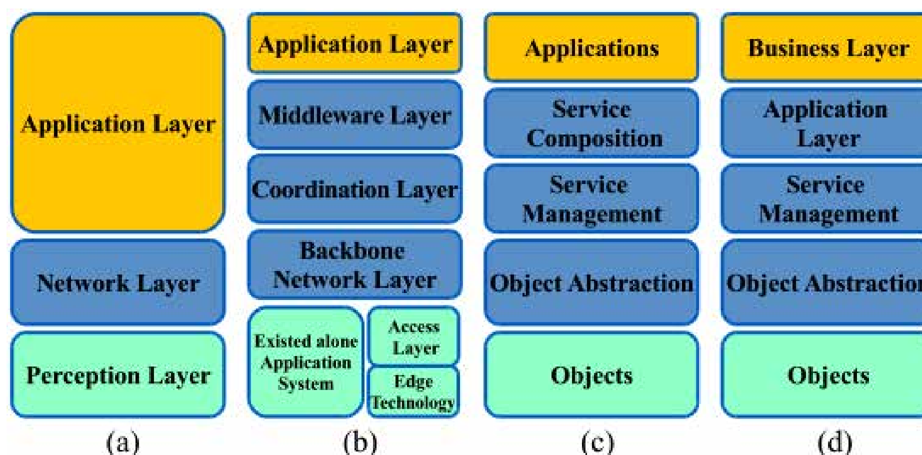


Рис. 1.3 – Варіанти архітектур IoT-систем: трирівнева, middleware-, SOA- та п’ятирівнева модель (модифіковано за Al-Fuqaha et al., 2015 [2]).

Одним із найважливіших напрямів подальших досліджень стало перенесення обчислювальних ресурсів ближче до джерел даних - на рівень периферійних вузлів. Концепція edge computing, розроблена Shi et al. (2016), базується на принципі децентралізованої аналітики, коли частина задач оброблення, агрегації та зберігання даних виконується локально на edge-рівні, а хмарні сервіси забезпечують довготривале навчання моделей і централізовану координацію (рис. 1.4).

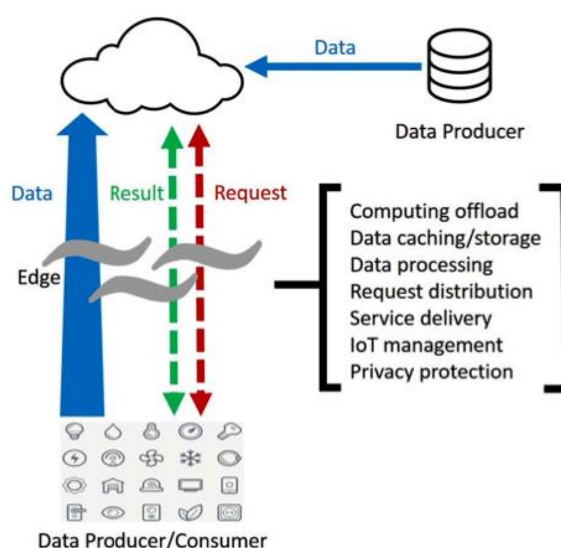


Рис. 1.4 – Концептуальна парадигма edge-обчислень у системах інтернету речей (модифіковано за Shi et al., 2016 [3]).

Gubbi et al. (2013) розширили базову архітектуру IoT-систем, запропонувавши концептуальну модель взаємодії сенсорів, мережевої інфраструктури, хмарних обчислень та прикладних сервісів. У центрі цієї моделі розташована хмара як уніфікований обчислювальний вузол, який об’єднує фізичний рівень сприйняття, комунікаційні канали та бізнес-рівень прийняття рішень (рис. 1.5). Саме ця архітектура покладена в основу більшості сучасних IoT-платформ з інтелектуальними алгоритмами аналізу потокових даних.

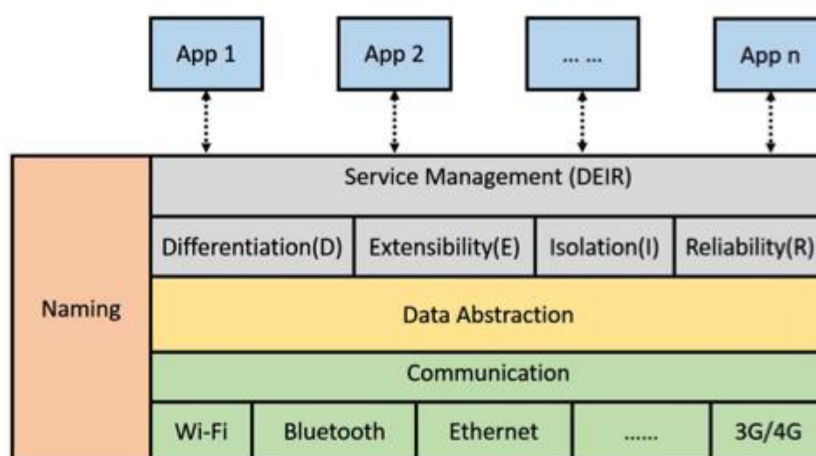


Рис. 1.5 – Концептуальна модель інтеграції хмарних сервісів у структурі IoT-систем (адаптовано за Gubbi et al., 2013 [4]).

Водночас Національний інститут стандартів і технологій США (NIST) у звіті SP 800-183 (2016) розробив узагальнену модель “Networks of Things” (NoT), яка розглядає інтернет речей як сукупність автономних кластерів сенсорів, агрегаторів і сервісів-утиліт, що обмінюються інформацією через комунікаційні канали. Цей підхід (рис. 1.6) відображає логічну еволюцію від простої багаторівневої архітектури до самоорганізованих динамічних мереж, здатних самостійно масштабуватися й адаптуватися до зміни навантажень.

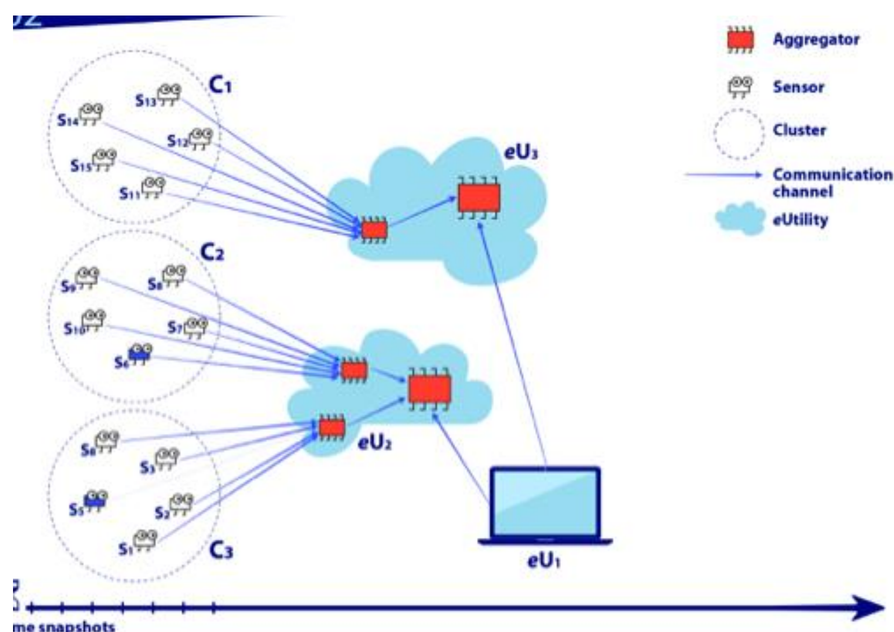


Рис. 1.6 – Модель «мережі речей» із кластеризацією сенсорів та агрегаторів (адаптовано за NIST SP 800-183, 2016 [5]).

На основі узагальнених моделей, поданих у працях [1–5], у даній роботі буде розроблено та досліджено комп'ютерну систему збору, оброблення та інтелектуального аналізу IoT-даних, що поєднує принципи SOA-модульності, розподілених edge-обчислень і адаптивних алгоритмів машинного навчання.

Наукова новизна полягає у синтезі архітектури, яка забезпечує динамічний розподіл обчислювальних навантажень між польовими пристроями та серверною аналітикою, використанні інтелектуальних моделей прогнозування станів середовища на основі часових рядів, а також у побудові інформаційної інфраструктури, що підтримує автоматизовану маршрутизацію потоків телеметрії через MQTT/HTTPS-канали з мінімальною затримкою.

Отримані результати стануть основою для подальшого проектування архітектури системи, алгоритмів її функціональних модулів і розроблення програмного забезпечення, яке реалізує повний цикл життєдіяльності даних - від сенсорного рівня до інтелектуальної аналітики та прийняття керуючих рішень у реальному часі.

1.3 Аналіз існуючих рішень

Розвиток сучасних комп’ютерних систем з IoT-пристроями тісно пов’язаний із впровадженням інтегрованих платформ, які поєднують сенсорну інфраструктуру, протоколи передавання даних, аналітику великих обсягів інформації та алгоритми машинного навчання. Для обґрунтування архітектури досліджуваної системи було проаналізовано п’ять провідних рішень, які визначають тенденції у сфері IoT-аналітики: Google Cloud IoT Core, AWS IoT Analytics / Greengrass, Microsoft Azure IoT Hub, Siemens MindSphere та Node-RED + TensorFlow Lite.

Перше рішення - Google Cloud IoT Core - забезпечує безпечне підключення тисяч пристроїв, централізовану маршрутизацію даних і інтеграцію з аналітичними сервісами BigQuery та Cloud ML. Архітектура системи (рис. 1.7) складається з рівнів збору телеметрії, обробки в хмарі та використання результатів аналітики. Сервіс підтримує двосторонню комунікацію за протоколами MQTT та HTTPS, що дозволяє реалізувати адаптивні сценарії керування пристроями на основі прогнозних моделей.

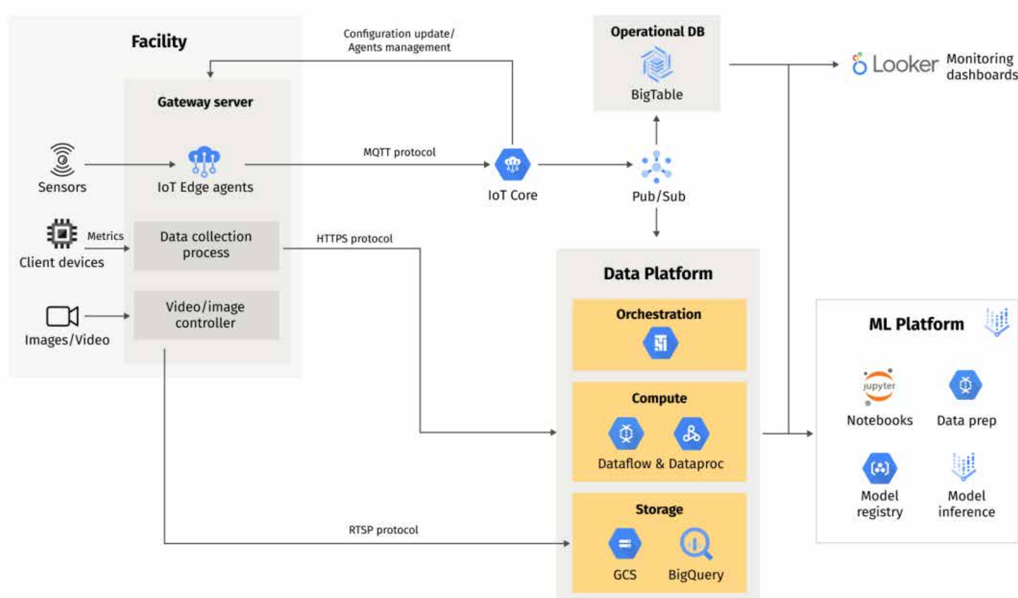


Рис. 1.7 – Архітектура Google Cloud IoT Core із потоковою аналітикою та машинним навчанням (адаптовано за Google LLC, 2023).

Друге рішення AWS IoT Analytics / AWS Greengrass - поєднує хмарну аналітику з периферійною обробкою даних. Як показано на рис. 1.8, периферійні вузли (Greengrass) здійснюють фільтрацію та попередню агрегацію даних перед їх передачею в аналітичний модуль AWS IoT Analytics. Система також підтримує інтеграцію з Amazon QuickSight для побудови візуальних дашбордів і з Lambda для реалізації автоматичних тригерів на основі подій. Така архітектура дає змогу зменшити затримки, оптимізувати трафік і реалізувати сценарії локальної автономії пристроїв.

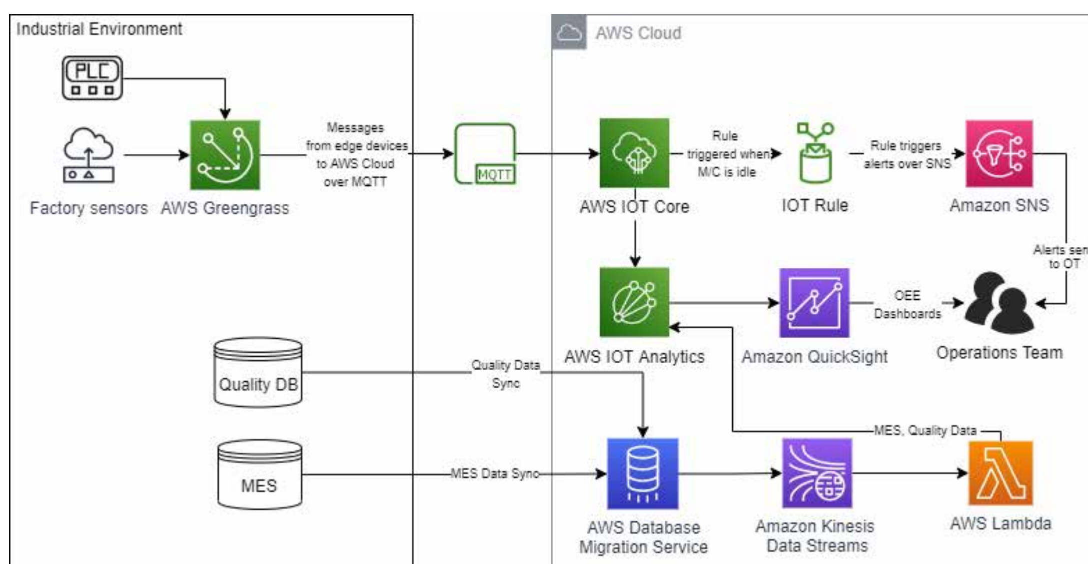


Рис. 1.8 – Система AWS IoT Analytics / Greengrass для гібридної обробки даних (адаптовано за Amazon Web Services, 2023).

Третє рішення Microsoft Azure IoT Hub - орієнтоване на промислове використання та комплексне управління сенсорними пристроями. Як показано на рис. 1.9, архітектура складається з шарів потокового введення (Event Hub, Kafka), середовища обробки (Azure Functions, Stream Analytics) і рівня аналітики (Azure Machine Learning / Power BI). Система забезпечує масштабування, резервування та динамічне оновлення моделей машинного навчання безпосередньо на edge-вузлах, що особливо важливо для високонавантажених середовищ реального часу.

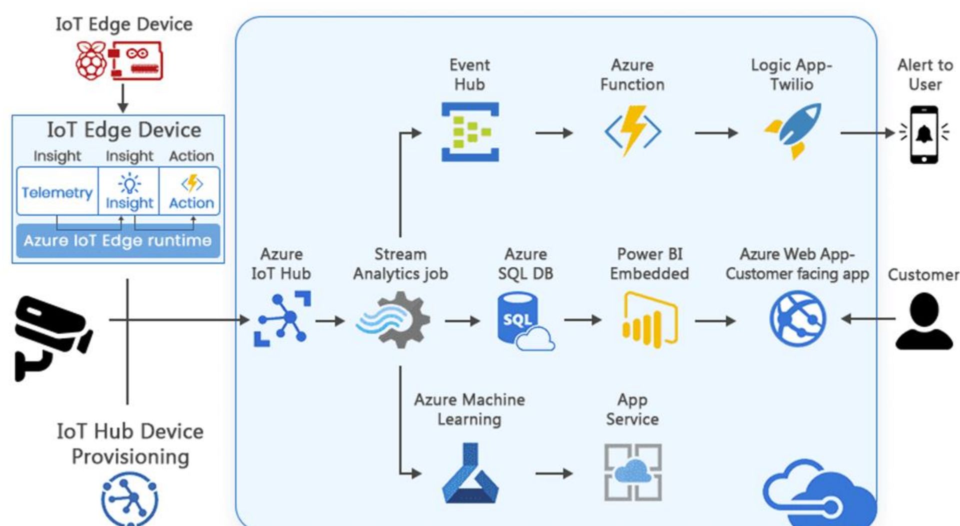


Рис. 1.9 – Інтегрована архітектура Microsoft Azure IoT Hub із компонентами аналітики ML (адаптовано за Microsoft Corporation, 2023).

Четверте рішення Siemens MindSphere - є промисловою платформою для моніторингу, діагностики та оптимізації виробничого обладнання. Як зображено на рис. 1.10, структура системи включає рівні «Machine», «Industrial Edge» та «Cloud», які взаємодіють через модулі управління пристроями, логування, моніторинг та користувацькі додатки. Хмарна частина забезпечує аналітику та управління життєвим циклом активів, тоді як edge-компоненти виконують локальну обробку й контроль технологічних параметрів.

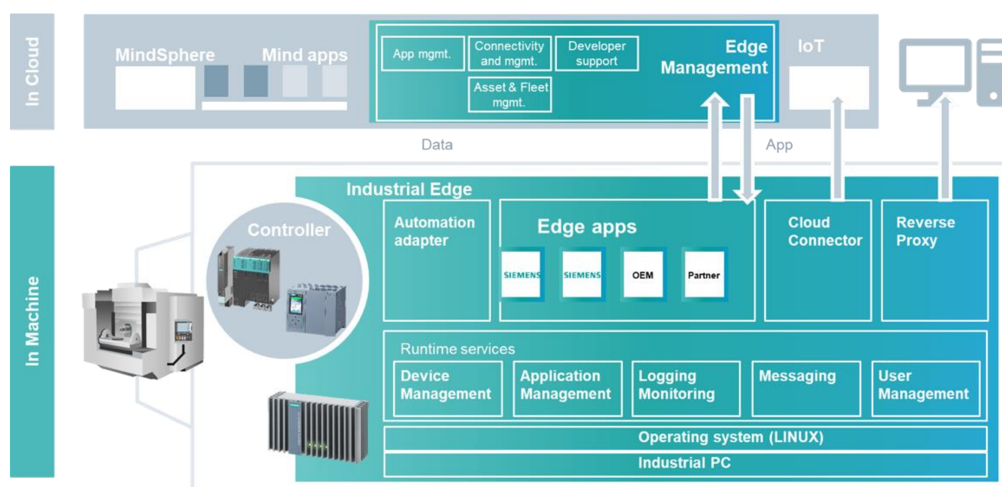


Рис. 1.10 – Промислова IoT-платформа Siemens MindSphere з інтеграцією Industrial Edge (адаптовано за Siemens AG, 2023).

Порівняльний аналіз показує, що провідні платформи реалізують спільні концепції - розподілення обчислень між периферійним і хмарним рівнями,

підтримку MQTT/HTTPS-комунікацій та впровадження ML-модулів. Водночас жодне з рішень не поєднує **а**даптивну аналітику, локальне прогнозування та відкриту Python-інфраструктуру в єдиному середовищі, орієнтованому на дослідження інтелектуальних алгоритмів для систем з IoT-пристроями.

Тому в даній роботі буде розроблено власну комп'ютерну систему збору, оброблення й прогнозування IoT-даних, побудовану на Python із використанням бібліотек Scikit-learn, Pandas, NumPy, PyQt6 та брокера Mosquitto для обміну повідомленнями, що дозволить об'єднати переваги edge-аналітики, машинного навчання та адаптивного управління. Порівняльна характеристика представлена у таблиці 1.2.

Таблиця 1.2 – Порівняльна характеристика існуючих IoT-платформ і запропонованої системи

№	Система	Тип архітектури	Основні функції	Використані технології	Особливості
1	Google Cloud IoT Core	Хмарна	Збір даних, аналітика ML	MQTT, BigQuery, Cloud ML	Висока масштабованість, відсутність edge-аналітики
2	AWS IoT Analytics / Greengrass	Гібридна (edge + cloud)	Локальна обробка, потокова аналітика	Lambda, Kinesis, S3	Висока складність інтеграції
3	Microsoft Azure IoT Hub	Мікросервісна	Керування пристроями, ML-модулі	AMQP, Azure ML	Пропрієтарна екосистема
4	Siemens MindSphere	Хмарно-edge	Моніторинг, прогноз стану обладнання	OPC UA, Python	Висока вартість впровадження
5	Node-RED + TensorFlow Lite	Відкрита	Візуальне програмування, локальне ML	Python, MQTT	Обмежена масштабованість
6	Запропонована система	Edge + Cloud (на Python)	Збір, аналітика, прогноз у реальному часі	Python (PyQt6, Scikit-learn, Pandas), MQTT	Відкрите середовище, адаптивні алгоритми

У результаті аналізу визначено, що запропонована система поєднує гнучкість open-source-технологій і інтелектуальну аналітику IoT-потоків, забезпечуючи багаторівневу обробку даних та високу адаптивність до зміни умов середовища. Отримані висновки лягли в основу проектування архітектури системи, функціональної схеми та алгоритмічних модулів, що реалізують автоматизоване прогнозування параметрів у реальному часі.

1.4 Функціональна структура та принципи роботи системи

Функціональна структура розробленої комп'ютерної системи базується на принципах інтелектуальної обробки IoT-даних у реальному часі, що поєднує апаратні вузли збору інформації, модулі попередньої аналітики на периферії (edge), брокер повідомлень та хмарні сервіси машинного навчання. Система побудована таким чином, щоб забезпечити замкнений цикл оброблення даних - від сенсорного рівня до прийняття рішень і автоматизованого зворотного впливу на об'єкти управління.

Основна ідея архітектури полягає у розподіленні обчислювальних завдань між крайовими пристроями та хмарною аналітикою залежно від складності обробки, затримки сигналу й пріоритетності подій. Це дозволяє зменшити навантаження на центральні сервери, скоротити час реакції системи та забезпечити безперервну роботу навіть за нестабільного з'єднання. Зокрема, на периферії реалізовано механізми кешування, фільтрації та попереднього аналізу телеметрії, а в хмарній частині - інструменти потокової обробки, навчання моделей і прогнозування поведінки системи.

На рис.1.11 подано узагальнену функціональну схему комп'ютерної системи, що демонструє логічну взаємодію основних підсистем. У лівій частині зображено польовий рівень - сенсорні вузли, крайові контролери та модулі Edge AI, які здійснюють локальну аналітику та контроль акторів. Далі показано транспортно-комунікаційну частину з брокером повідомлень (MQTT / Kafka),

який забезпечує надійну маршрутизацію потоків даних. Центральна частина схеми представляє хмарне сховище та аналітичні модулі: Data Lake / DWH, ML Pipeline, Feature Store і Realtime Inference, що формують основу інтелектуальної обробки. Праворуч подано рівень операцій і взаємодії з користувачем - веб/мобільний клієнт, API Gateway, аналітичну панель, моніторинг і систему сповіщень**.**

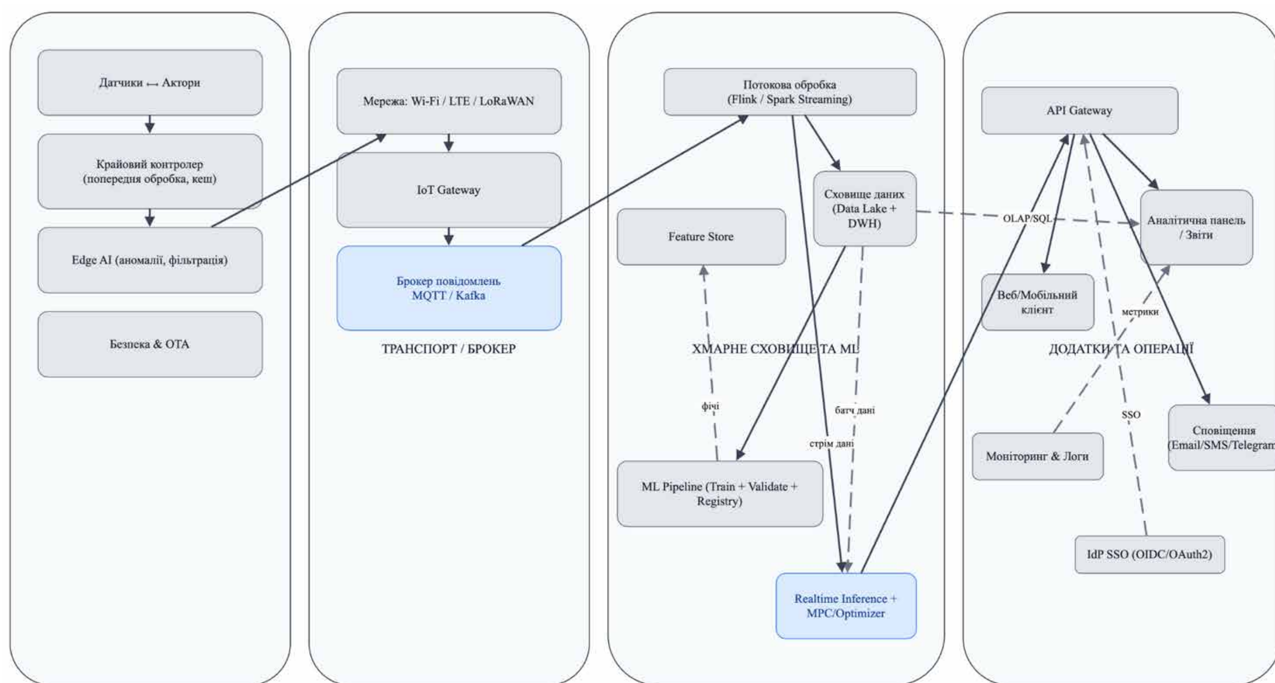


Рис. 1.11 – Функціональна структура комп’ютерної системи з інтелектуальними алгоритмами для IoT-пристроїв (авторська розробка).

Обрана архітектура відповідає сучасним тенденціям розвитку edge–cloud інтеграції та реалізує принцип intelligent feedback loop, коли результати аналітики безпосередньо впливають на роботу сенсорних вузлів і керуючих пристроїв. Така побудова дозволяє реалізувати сценарії самонавчання системи**,** де моделі машинного навчання постійно оновлюються на основі актуальних даних і зберігаються у Feature Store для подальшого використання.

Запропонована структура забезпечує наукову новизну через інтеграцію адаптивних алгоритмів ML у процеси керування IoT-пристроями з використанням Python-технологій, зокрема Scikit-learn, Pandas, FastAPI, PyQt6**.** Вона поєднує точність хмарної аналітики з швидкодією edge-обробки, що дає можливість виконувати прогнозування, оптимізацію та виявлення аномалій у

режимі реального часу. У подальших розділах на основі цієї функціональної моделі буде здійснено алгоритмізацію основних модулів системи та обґрунтовано принципи їх програмної реалізації.

Таблиця 1.3 – Основні функціональні модулі комп'ютерної системи

№	Функціональний модуль	Призначення	Ключові технології	Очікуваний результат
1	Sensor/Edge Layer	Збір та локальна обробка даних, фільтрація шумів	ESP32, Python Edge AI	Зменшення обсягу переданих даних, виявлення аномалій
2	Transport/Broker Layer	Маршрутизація повідомлень між компонентами	MQTT, Kafka, TLS	Надійна передача даних у реальному часі
3	Cloud Data & ML Layer	Навчання моделей, аналітика, зберігання даних	Spark, Flink, Scikit-learn	Побудова моделей прогнозування, аналітика потоків
4	Realtime Inference Layer	Прогнозування, оптимізація, MPC-регулювання	Python ML, REST API	Автоматичне прийняття рішень у реальному часі
5	Application Layer	Візуалізація, керування, моніторинг, сповіщення	FastAPI, PyQt6, HTML5	Зручна аналітика й управління системою користувачем

Функціональна структура системи є інтегрованою моделлю IoT-аналітики, що поєднує апаратні, програмні та аналітичні компоненти в єдиному середовищі. Вона реалізує адаптивну багаторівневу обробку даних, забезпечуючи автономність периферійних пристроїв і глибоку аналітику в хмарі. Запропоноване рішення поєднує точність моделей машинного навчання з ефективністю розподілених обчислень і створює технологічну основу для впровадження інтелектуальних систем у галузях моніторингу, енергетики, агроаналізу та автоматизованого управління середовищами з великою кількістю IoT-вузлів.

1.5 Аналіз вимог системи

Для забезпечення коректної розробки та впровадження комп’ютерної системи з інтелектуальними алгоритмами для IoT-пристроїв проведено системний аналіз вимог, що визначає функціональні, технічні та безпекові параметри майбутнього рішення. Аналіз побудований з урахуванням особливостей розподілених IoT-середовищ, потокової обробки даних та застосування моделей машинного навчання для прогнозування і прийняття рішень у реальному часі.

Розроблювана система орієнтована на роботу з численними периферійними вузлами (сенсорами, контролерами, акторними модулями), які передають телеметрію через MQTT-брокер у серверну частину для подальшої обробки. Основними вимогами є висока масштабованість, стійкість до відмов, захищеність даних і мінімальні затримки при комунікації.

Система має забезпечувати виконання повного циклу операцій – від збору даних до їх аналізу та реакції на виявлені аномалії або зміни параметрів середовища. До основних функцій належать: моніторинг стану IoT-вузлів, збереження телеметрії, виконання ML-аналізу, візуалізація результатів та формування звітів для користувача. У таблиці 1.4 наведено узагальнені функціональні вимоги системи.

Таблиця 1.4 – Функціональні вимоги комп’ютерної системи

№	Вимога	Опис	Очікуваний результат
1	Збір і передача телеметрії	Отримання даних від датчиків через MQTT/TLS	Безперервний потік даних у реальному часі
2	Попередня обробка даних	Фільтрація, нормалізація, виявлення аномалій	Зменшення шуму, підвищення достовірності
3	Збереження даних	Автоматичне записування у БД PostgreSQL / Data Lake	Гарантована цілісність і доступність даних
4	Машинне навчання	Навчання моделей на основі історичних даних	Отримання моделей прогнозування параметрів
5	Аналітичні звіти	Генерація HTML- та PDF-зведень	Інформаційна підтримка прийняття рішень

Продовження таблиці 1.4

6	Користувацький інтерфейс	Графічний або веб-інтерфейс (PyQt6 / Web)	Зручне керування та моніторинг системи
7	Інтеграція з API	Підключення зовнішніх сервісів (погода, повідомлення)	Розширення функціональності системи

До технічних характеристик належать параметри продуктивності, доступності та апаратної сумісності. Оскільки система розподілена, важливими є підтримка протоколів Wi-Fi, Ethernet, MQTT/TLS, а також автономність крайових вузлів. На рівні сервера реалізується асинхронна обробка запитів, потокове з'єднання з брокером і аналітичні обчислення у Python-середовищі.

У табл. 1.5 подано основні технічні вимоги системи.

Таблиця 1.5 – Технічні вимоги до системи

№	Показник	Вимога / межа	Коментар
1	Час реакції на подію	≤ 200 мс	Включно з обробкою MQTT-повідомлення
2	Гарантія доставки даних	≥ 99.9 %	Використання QoS 2 та TLS
3	Масштабованість	≥ 1000 вузлів	Горизонтальне масштабування брокера
4	Доступність сервісу	≥ 99.5 %	Резервування і відновлення після збоїв
5	Формат даних	JSON / CSV / SQL	Єдиний формат для аналітики
6	Обсяг локального кешу	≥ 512 MB	Забезпечує автономність edge-вузлів
7	Підтримка ОС	Linux / Windows / Raspberry Pi OS	Універсальність розгортання системи

Безпека є критичною складовою проєкту, оскільки система працює з потоками даних, які можуть містити конфіденційну інформацію. Передбачено застосування TLS-шифрування, ролей користувачів (RBAC), аутентифікації через OIDC/OAuth2 та централізованого журналювання подій. Також реалізується моніторинг цілісності даних та виявлення спроб несанкціонованого

доступу. У табл. 1.6 наведено узагальнення вимог до інформаційної безпеки системи.

Таблиця 1.6 – Вимоги до інформаційної безпеки системи

№	Вимога	Реалізація	Призначення
1	Аутентифікація користувачів	OIDC / OAuth2, токени доступу	Обмеження доступу до API
2	Авторизація за ролями	RBAC-механізм у серверній частині	Розмежування прав користувачів
3	Шифрування даних	TLS 1.3 / HTTPS	Захист трафіку між вузлами
4	Моніторинг подій	Централізований журнал (Logs DB)	Виявлення інцидентів безпеки
5	Резервне копіювання	Автоматичний backup БД	Відновлення після збоїв
6	Контроль цілісності	Хешування SHA-256	Гарантія незмінності даних
7	Захист мережі	VPN / Firewall	Ізоляція внутрішнього трафіку IoT

Проведений аналіз вимог дозволяє сформувати повну специфікацію системи, що охоплює всі рівні - від апаратного до аналітичного. Запропоновані вимоги визначають архітектурні рішення, програмну реалізацію та принципи безпеки, необхідні для стабільної роботи інтелектуальної IoT-платформи.

Виконання зазначених вимог забезпечить високу надійність, гнучкість і масштабованість системи, а також можливість інтеграції алгоритмів машинного навчання без втрати швидкодії. На основі цих вимог у подальших розділах буде виконано алгоритмізацію основних модулів і створення програмного забезпечення системи, реалізованого в середовищі Python з використанням технологій FastAPI, Scikit-learn, Pandas, PyQt6 та брокера Mosquitto.

1.6 Постановка завдання

На основі аналізу предметної області, існуючих рішень та визначених вимог сформульовано завдання створення комп’ютерної системи з

інтелектуальними алгоритмами для систем з IoT-пристроями, що забезпечує автоматизований збір, обробку, прогнозування та візуалізацію даних у реальному часі. Система має реалізувати повний цикл оброблення інформації — від сенсорного рівня до прийняття керуючих рішень, поєднуючи можливості edge-аналітики та хмарного машинного навчання.

Основна мета полягає у розробленні архітектури, здатної до адаптивного розподілу обчислень між периферійними пристроями та серверною аналітикою залежно від обсягу даних, затримки та пріоритетності завдань. На периферії виконуються фільтрація та локальний аналіз телеметрії, а у хмарній частині - навчання моделей, прогнозування параметрів і формування рішень для керування виконавчими пристроями.

Система повинна функціонувати за принципом замкненого інтелектуального контуру (intelligent feedback loop): зібрані сенсорами дані обробляються в реальному часі, результати прогнозування повертаються до керуючих вузлів, що забезпечує адаптивне реагування на зміни середовища. Для цього застосовуються технології Python (Scikit-learn, Pandas, NumPy), MQTT-брокер Mosquitto, PostgreSQL для зберігання даних та FastAPI / PyQt6 для користувацької взаємодії.

Поставлене завдання передбачає:

– створення моделі обміну даними між сенсорними вузлами, брокером і аналітичним сервером;

– реалізацію модулів збору, попередньої обробки та прогнозування даних;

– інтеграцію алгоритмів машинного навчання для виявлення аномалій і трендів;

– побудову інтерфейсу для моніторингу стану системи та аналізу результатів.

Реалізація проєкту дозволить створити адаптивну інтелектуальну IoT-систему, яка поєднує автономність, масштабованість і здатність до самонавчання. Отриманий прототип стане основою для подальшого

вдосконалення методів інтелектуальної обробки даних і практичного застосування в енергетичних, екологічних та виробничих IoT-середовищах.

2 ПРОЄКТУВАННЯ ЕМУЛЯТОРА СЕРВЕРНОЇ ОБРОБКИ ПОДІЙ У КОМП'ЮТЕРНІЙ СИСТЕМІ З ІНТЕЛЕКТУАЛЬНИМИ АЛГОРИТМАМИ ДЛЯ IoT-ПРИСТРОЇВ

2.1 Принципова схема емулятора та структура каналів передачі подій

У межах розроблення комп'ютерної системи з інтелектуальними алгоритмами для IoT-пристроїв було створено емулятор сенсорного середовища, який відтворює роботу реальних пристроїв збору телеметрії. Його головна мета полягає у забезпеченні нормалізованого та контрольованого потоку подій, що надходять до аналітичного ядра системи через багаторівневу структуру обміну повідомленнями. Це дає змогу моделювати поведінку IoT-мереж без потреби у фізичному розгортанні великої кількості сенсорів і водночас досліджувати ефективність алгоритмів машинного навчання для прогнозування параметрів середовища.

На рис. 2.1 подано узагальнену принципову схему емулятора та структуру каналів передачі подій, яка відображає взаємодію між компонентами потокової обробки, сховищем даних та аналітичними модулями.

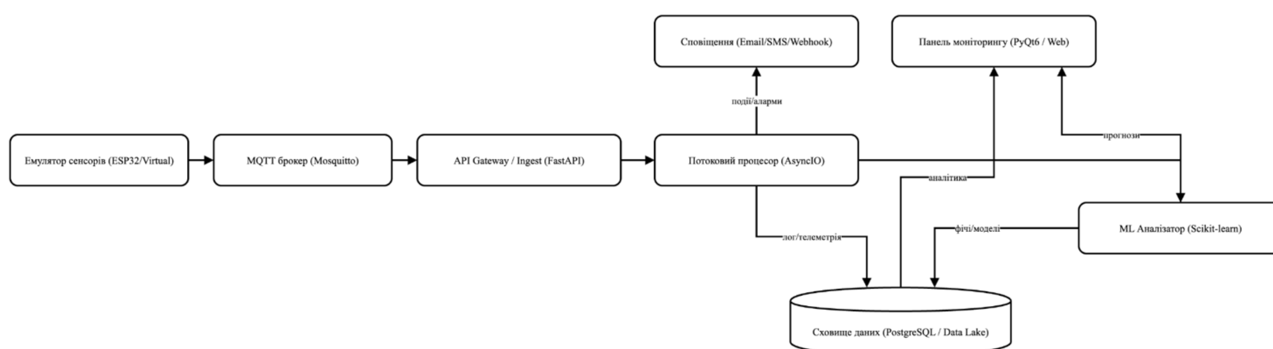


Рис. 2.1 – Принципова схема емулятора сенсорів та структура каналів передачі подій у комп'ютерній системі

Запропонована архітектура передбачає мультиканальну маршрутизацію телеметрії через MQTT-брокер, який виконує роль асинхронного буфера для обміну повідомленнями між периферією, API-шлюзом та аналітичним сервером.

Такий підхід забезпечує стабільність при високій частоті подій і дозволяє масштабувати систему за принципом publish/subscribe без зміни логіки ядра. Потоковий процесор формує єдиний простір подій, у якому відбувається агрегація, класифікація, нормалізація та передача даних до сховища або ML-аналізатора. Застосування технологій AsyncIO, FastAPI та Scikit-learn дозволяє реалізувати паралельну обробку потоків, зберігаючи часову синхронізацію і мінімізуючи затримку обчислень.

Структура каналів передачі подій узагальнена у табл. 2.1, де наведено основні типи потоків, їх призначення та протоколи комунікації.

Таблиця 2.1 – Основні канали передачі подій у системі

№	Канал подій	Призначення	Протокол / Технологія	Характеристики
1	Сенсорний потік	Передача телеметрії з емулятора ESP32	MQTT / TLS 1.3	Частота – 1 Гц, QoS = 2
2	Потік аналітики	Взаємодія потокового процесора з ML-модулем	REST / AsyncIO	Асинхронний обмін JSON
3	Потік логів	Запис телеметрії до PostgreSQL / Data Lake	SQL / Batch Write	Нормалізований запис
4	Потік подій/сповіщень	Формування повідомлень Email / Webhook	HTTP / SMTP	Реакція на тригери аномалій
5	Потік візуалізації	Передача прогнозів до панелі моніторингу	WebSocket / PyQt6	Затримка < 200 мс

Принциповим рішенням при проектуванні стало забезпечення інформаційної узгодженості між усіма каналами через уніфікований формат даних JSON і нормалізовану схему запису до бази даних. Це дозволяє усунути дублювання телеметрії, мінімізувати конфлікти при одночасних оновленнях і забезпечити точність подальшої аналітики. Така структура є базою для реалізації адаптивного “intelligent feedback loop”, у якому потоки даних не лише реєструються, а й впливають на алгоритмічне керування системою.

У результаті реалізована схема забезпечує високу відмовостійкість, масштабованість і відтворюваність аналітичних експериментів, що є критично важливим для дослідження й оптимізації моделей машинного навчання у сфері IoT-аналітики.

2.2 Електрична та монтажна схема дослідного стенду емулятора

Для побудови дослідного стенду було використано набір компонентів, що забезпечують реалізацію повного циклу оброблення IoT-подій - від вимірювання фізичних параметрів до їхньої обробки, аналізу й візуалізації результатів. Обладнання обрано з урахуванням критеріїв енергоефективності, точності вимірювань, надійності комунікаційта сумісності між апаратними рівнями. Кожен елемент системи виконує специфічну функцію в контексті моделювання розподіленої сенсорної мережі та відображає принципи реальної IoT-архітектури.

На рис. 2.2 показано мікроконтролер ESP32 DevKit, який виступає центральним вузлом периферійного рівня, забезпечуючи збір і попередню обробку даних сенсорів. Він має двоядерний процесор 240 MHz, інтегровані модулі Wi-Fi і Bluetooth, підтримку інтерфейсів GPIO, I^2C та UART. Завдяки вбудованому Wi-Fi-модулю ESP32 здійснює бездротову передачу телеметрії за протоколом MQTT з використанням TLS-шифрування.

Рис. 2.2 – Мікроконтролер ESP32 DevKit – периферійний вузол збору даних

Другим ключовим елементом стенду є Raspberry Pi 4 B (рис. 2.3), який виконує роль брокера MQTT та центрального шлюзу системи. Цей одноплатний комп’ютер підтримує повноцінне середовище Linux (Raspberry Pi OS), працює під керуванням ARM Cortex-A72 1.5 GHz та має 4 GB RAM, що дає змогу реалізувати сервер Mosquitto для маршрутизації потоків даних, а також локальні сервіси FastAPI та PostgreSQL. Raspberry Pi підключається до мережі через Ethernet або Wi-Fi та формує логічний центр взаємодії між сенсорними вузлами та аналітичним рівнем.

Рис. 2.3 – Одноплатний комп’ютер Raspberry Pi 4 B – MQTT-брокер і шлюз даних

Для збору параметрів середовища використано два типи сенсорів, що реалізують багатоканальний збір телеметрії. На рис. 2.4 подано датчик DHT22, який вимірює температуру й відносну вологість повітря. Його цифровий вихід забезпечує точність ±0.5 °C для температури та ±2 % для вологості. Модуль підключається до ESP32 через лінію DATA з резистором підтягування R1 = 10 kΩ до 3.3 V, що нормалізує логічний рівень сигналу.

Рис. 2.4 – Датчик DHT22 для вимірювання температури та вологості

На рис. 2.5 наведено сенсор BMP280, який вимірює атмосферний тиск і температуру з високою точністю та передає дані через інтерфейс I²C (SDA, SCL). Для стабільності роботи використано резистори підтягування R2 і R3 по 4.7 kΩ, що гарантує узгодженість рівнів 3.3 V і усуває шум у цифрових каналах. Комбінація DHT22 та BMP280 забезпечує комплексне оцінювання мікрокліматичних умов.

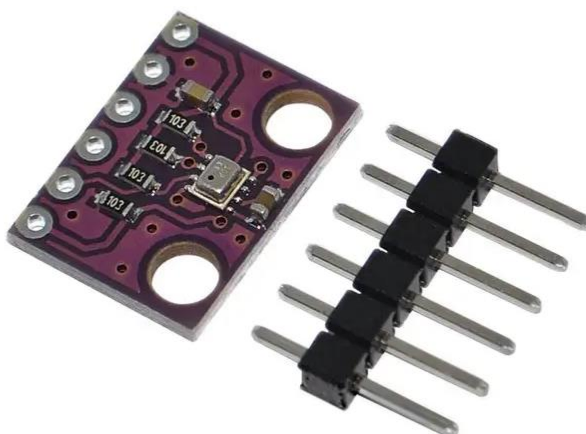


Рис. 2.5 – Сенсор BMP280 для вимірювання атмосферного тиску

Між ESP32 та Raspberry Pi використано логічний рівнеперетворювач (Level Shifter), зображений на рис. 2.6, який узгоджує сигнали 3.3 ↔ 5 V та

виключає перевантаження портів GPIO. Це рішення є критичним для надійної інтеграції мікроконтролера з периферією та мережевими пристроями, адже забезпечує електричну ізоляцію між високовольтною й низьковольтною логікою.

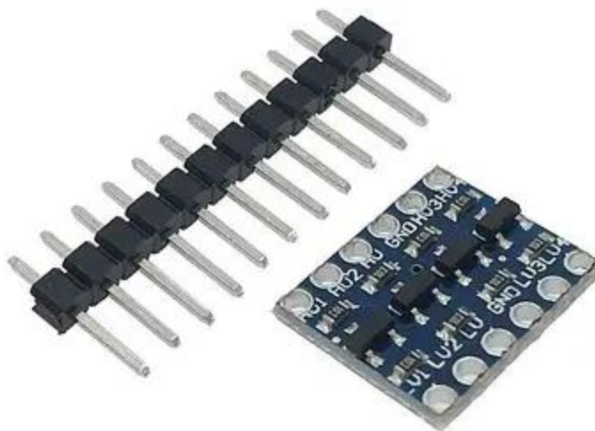


Рис. 2.6 – Модуль логічного рівнеперетворювача 3.3 ↔ 5 V

Живлення стенду забезпечується стабілізованим блоком живлення 5 V DC (≥ 3 A), який підтримує як Raspberry Pi, так і периферію ESP32 через макетну шину 3.3 V (рис. 2.7). Енергетична структура побудована за принципом розділення контурів навантаження, що запобігає просіданню напруги при імпульсному споживанні струму модулем Wi-Fi та гарантує сталість сигналів GPIO.

Рис. 2.7 – Блок живлення 5 V DC для живлення основних вузлів системи

Загальний перелік апаратних компонентів дослідного стенду наведено у табл. 2.2, де систематизовано основні технічні характеристики та функціональне призначення кожного елемента.

Таблиця 2.2 – Основні апаратні компоненти дослідного стенду

№	Компонент	Призначення	Основні параметри
1	ESP32 DevKit (MCU)	Центральний вузол збору телеметрії, передача MQTT-повідомлень, Wi-Fi TLS	3.3 V, Wi-Fi 2.4 GHz, 240 MHz
2	Raspberry Pi 4 B (MQTT Broker)	Приймання, зберігання та агрегація даних; керування потоками	5 V, 4 GB RAM, ARM Cortex-A72
3	DHT22	Вимірювання температури та вологості	$\Delta T \pm 0.5$ °C, $\Delta H \pm 2$ %
4	BMP280	Вимірювання атмосферного тиску та температури	$\Delta P \pm 1$ hPa, I^2C (0x76/0x77)
5	Level Shifter (3.3 ↔ 5 V)	Узгодження логічних рівнів між ESP32 та Raspberry Pi	MOSFET-модуль 4-канальний
6	LED + R = 220 Ω	Візуальна індикація стану або тривог	GPIO2, 3.3 V
7	БЖ 5 V DC (≥ 3 A)	Живлення системи, стабілізація напруги	5 V, 3 A мін.
8	Wi-Fi AP / Роутер	Забезпечення бездротової комунікації	IEEE 802.11 b/g/n
9	ПК (моніторинг / логування)	Аналіз, SSH-доступ, тестування MQTT-потоків	USB-UART, Ethernet

На рис. 2.8 наведено електричну схему дослідного стенду, що відображає логічні зв’язки між основними вузлами: мікроконтролером ESP32 DevKit, сенсорними модулями DHT22 і BMP280, блоком живлення 5 V DC, MQTT-брокером на Raspberry Pi 4B та зовнішніми каналами моніторингу. Схема побудована за принципами модульної взаємодії і функціональної декомпозиції, що дозволяє ізолювати фізичні шини від транспортних рівнів обміну даними. Для живлення цифрових компонентів використовується розділення на шину 5 V (силову) та шину 3.3 V (логічну). Такий підхід забезпечує стабільність роботи мікроконтролера і сенсорів навіть за змінного струмового навантаження від Wi-Fi-модуля.

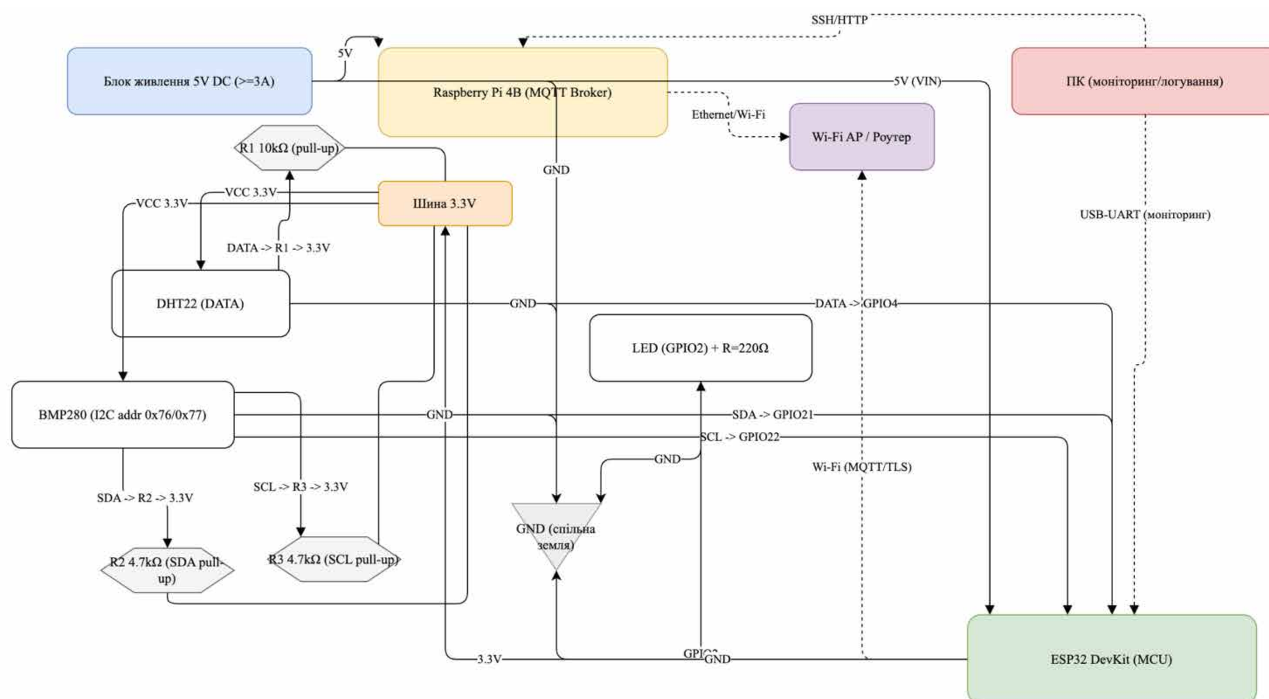


Рис. 2.8 – Електрична схема з’єднань апаратних компонентів дослідного стенду

У процесі проєктування електричних з’єднань було застосовано нормалізацію сигналів та узгодження рівнів логічної напруги. Лінії SDA та SCL від сенсора BMP280 підтягнуті до 3.3 V через резистори R2 та R3 номіналом 4.7 kΩ, що мінімізує вплив електромагнітних перешкод на I²C-інтерфейс. Для каналу DATA від DHT22 використано R1 = 10 kΩ («pull-up»), який нормалізує логічний рівень та усуває дрейф сигналу при переході з високого в низький стан. Така топологія гарантує електричну сумісність усіх елементів та підвищує точність передачі даних у динамічному режимі MQTT-обміну.

На рис. 2.9 подано монтажну схему стенду, що демонструє реальне фізичне розташування компонентів на макетній платі (Breadboard) з підключенням через кольорові шини живлення. Розподіл дротів виконано відповідно до стандарту кольорового кодування: червоний – 5 V, помаранчевий – 3.3 V, чорний – GND, зелений – SDA, синій – SCL, жовтий – DATA, пурпуровий – GPIO2→LED. Така візуальна сегментація спрощує обслуговування, зменшує ризик помилкового

підключення та дозволяє швидко ідентифікувати канали логічного взаємозв’язку.

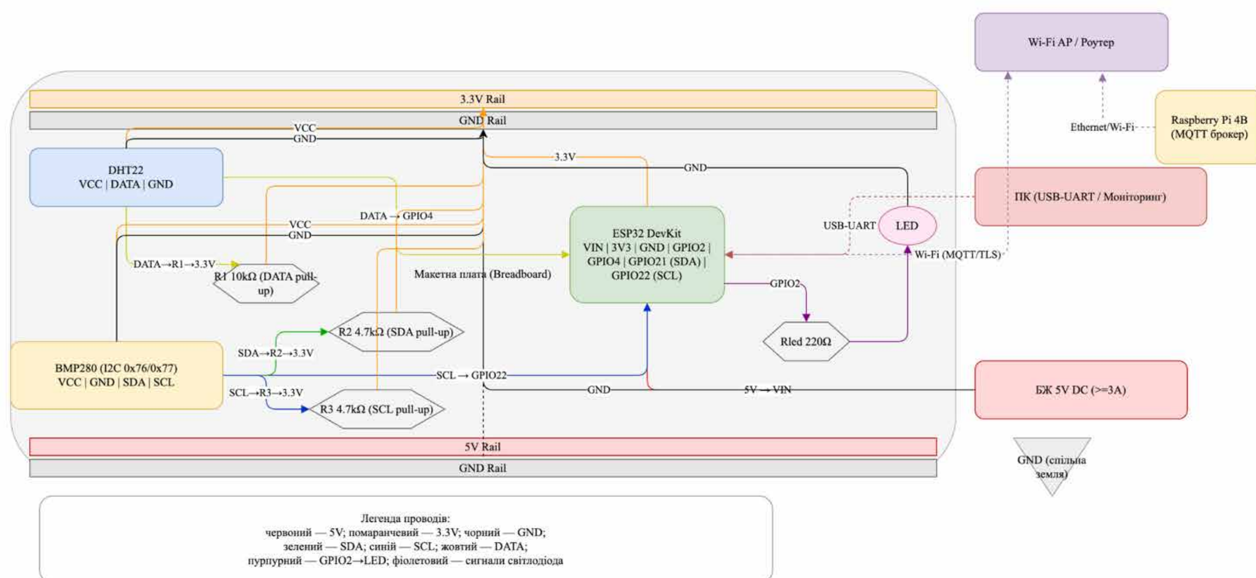


Рис. 2.9 – Монтажна схема розміщення компонентів на макетній платі

У табл. 2.3 наведено таблицю з основними електричними параметрами та з’єднаннями, що характеризують вхідні та вихідні лінії стенду, відповідно до норм електротехнічної сумісності та енергетичної ефективності.

Таблиця 2.3 – Електричні параметри з’єднань компонентів дослідного стенду

№	Лінія сигналу	Компонент-джерело	Компонент-приймач	Рівень напруги (V)	Номінал резистора	Призначення
1	DATA	DHT22	ESP32 GPIO4	3.3	R1 = 10 кΩ	Підтягування цифрового сигналу
2	SDA	BMP280	ESP32 GPIO21	3.3	R2 = 4.7 кΩ	Лінія даних I²C
3	SCL	BMP280	ESP32 GPIO22	3.3	R3 = 4.7 кΩ	Тактова лінія I²C
4	GPIO2	ESP32	LED + Rled 220 Ω	3.3	Rled = 220 Ω	Індикація активності
5	VIN	БЖ 5 V DC	ESP32 / Raspberry Pi 4B	5 / 3.3	—	Основне живлення вузлів

Продовження таблиці 2.3

6	GND	Всі модулі	Спільна земля	0	—	Нормалізація потенціалів

Додатково система включає Wi-Fi роутер, який забезпечує бездротовий транспорт MQTT-пакетів між ESP32 та Raspberry Pi 4B з використанням TLS-шифрування. Зв’язок між Raspberry Pi та ПК виконується через SSH/HTTP або USB-UART, що дозволяє проводити моніторинг, логування та віддалене адміністрування MQTT-брокера.

Загалом електрична та монтажна реалізація стенду ґрунтується на принципах нормалізації сигналів, мінімізації міжканальних перешкод, спільного контуру заземлення та енергетичного розділення шин. Це дозволило створити стабільну, масштабовану й відтворювану апаратну основу для дослідження роботи IoT-емулятора. Вибір модульної конфігурації з ESP32 та Raspberry Pi дозволяє не лише проводити експерименти з аналітичними алгоритмами, а й тестувати нові моделі мережевої взаємодії на рівні MQTT/TLS з гарантованою цілісністю та достовірністю даних.

2.3 Передумови створення програмного емулятора серверної логіки та вибір технологій

Розроблення програмного емулятора серверної логіки було обумовлено потребою у відтворюваному та масштабованому середовищі тестування, яке б дозволяло перевіряти коректність обміну даними між мікроконтролером ESP32, сенсорними модулями та серверною частиною системи без використання реальної інфраструктури. Емулятор забезпечує симуляцію процесів публікації, маршрутизації та обробки MQTT-повідомлень, а також імітує поведінку аналітичного ядра, яке обробляє телеметричні дані в режимі реального часу. Основною передумовою створення стала необхідність нормалізації середовища

випробувань для алгоритмів збору, фільтрації та передачі даних у межах IoT-архітектури, що унеможливлює вплив апаратних обмежень на якість досліджень.

Вибір технологій визначався принципами кросплатформеності, асинхронності та безпеки. Для реалізації серверної частини обрано мову Python з використанням фреймворку FastAPI, який забезпечує неблокувальну обробку запитів і сумісність з клієнтами MQTT-протоколу. Брокер повідомлень реалізовано за допомогою Eclipse Mosquitto, що підтримує TLS-шифрування, QoS-рівні 0–2 та інтеграцію з SQLite або PostgreSQL для зберігання історії телеметрії. Для візуалізації та відлагодження даних застосовується MQTT Explorer і серверна панель Grafana, яка дозволяє відстежувати параметри, передані сенсорами DHT22 і BMP280.

На рівні взаємодії між компонентами використовується асинхронна модель обміну повідомленнями, що ґрунтується на принципі "publish/subscribe". Така структура дозволяє створити незалежні модулі для публікації сенсорних даних, обробки аналітичних подій і моніторингу результатів. Серверний емулятор також підтримує REST-інтерфейси, через які аналітична система може запитувати накопичену інформацію, виконувати агрегацію та формувати звіти.

У табл. 2.4 наведено вибрані технології та інструменти, що використовуються для реалізації програмного середовища емуляції.

Таблиця 2.4 – Вибір технологій для реалізації програмного емулятора

№	Компонент	Технологія / бібліотека	Призначення
1	Серверна логіка	Python 3.11 + FastAPI	Реалізація REST-інтерфейсів та асинхронної обробки подій
2	MQTT-брокер	Eclipse Mosquitto	Передача даних між ESP32 і сервером за протоколом MQTT/TLS
3	База даних	PostgreSQL / SQLite	Збереження історії сенсорних даних і журналів повідомлень
4	Моніторинг	Grafana, MQTT Explorer	Візуалізація параметрів і перевірка обміну даними
5	Контейнеризація	Docker	Ізоляція серверного середовища, забезпечення переносимості

Продовження таблиці 2.4

6	Безпека	TLS 1.3, certbot (SSL)	Шифрування з’єднань і автентифікація клієнтів

Результатом впровадження такої технологічної конфігурації стало створення повноцінного віртуального середовища, що здатне імітувати роботу реальної IoT-мережі. Емулятор серверної логіки забезпечує масштабовану взаємодію компонентів, дозволяє проводити тестування на різних QoS-рівнях і досліджувати поведінку системи під навантаженням без ризику апаратних збоїв. Завдяки цьому підхід гарантує надійність, повторюваність і відтворюваність експериментів, що є критично важливим для аналітичних досліджень у галузі інтернету речей.

2.4 Моделювання предметної області емулятора серверної обробки подій

Моделювання предметної області емулятора серверної обробки подій спрямовано на формалізацію функціональної структури системи, визначення взаємодії між користувачами, процесами та сервісами, а також на опис потоків даних у межах логічної архітектури IoT-емулятора. Метою моделювання є забезпечення однозначного уявлення про логіку серверної частини, взаємозв’язок підсистем і сценарії виконання подій, що дозволяє провести узгодження між аналітичними, апаратними та програмними рівнями системи.

На рис. 2.10 подано UML-діаграму прецедентів, яка відображає ролі основних користувачів і зовнішніх акторів, а також ключові сценарії взаємодії з емульованим середовищем. У моделі виділено чотири основні категорії користувачів:

1. інженер-тестувальник - ініціює процеси конфігурації, налаштування MQTT-тем і запуску симуляції подій;

2. Оператор системи - переглядає дашборди, генерує звіти, контролює потоки даних і стан аналітичних моделей;

3. Аналітик даних - відповідає за оновлення ML-моделей, виявлення аномалій і оцінку ефективності прогнозів;

4. Адміністратор системи - виконує керування користувачами, архівацію даних і налаштування політик доступу (RBAC).

Емулятор обробляє події, які надходять через MQTT-брокер від периферійних пристроїв (ESP32) та зовнішніх сервісів. Усередині системи реалізовано прецеденти "Прийом повідомлень MQTT", "Обробка подій", "Розрахунок ознак", "Інференс моделі ML", "Виявлення аномалій", "Запис телеметрії у БД", "Надсилання сповіщень" і "Архів/Бекап бази даних", що відповідають фазам життєвого циклу повідомлення - від отримання до аналітичної реакції системи.

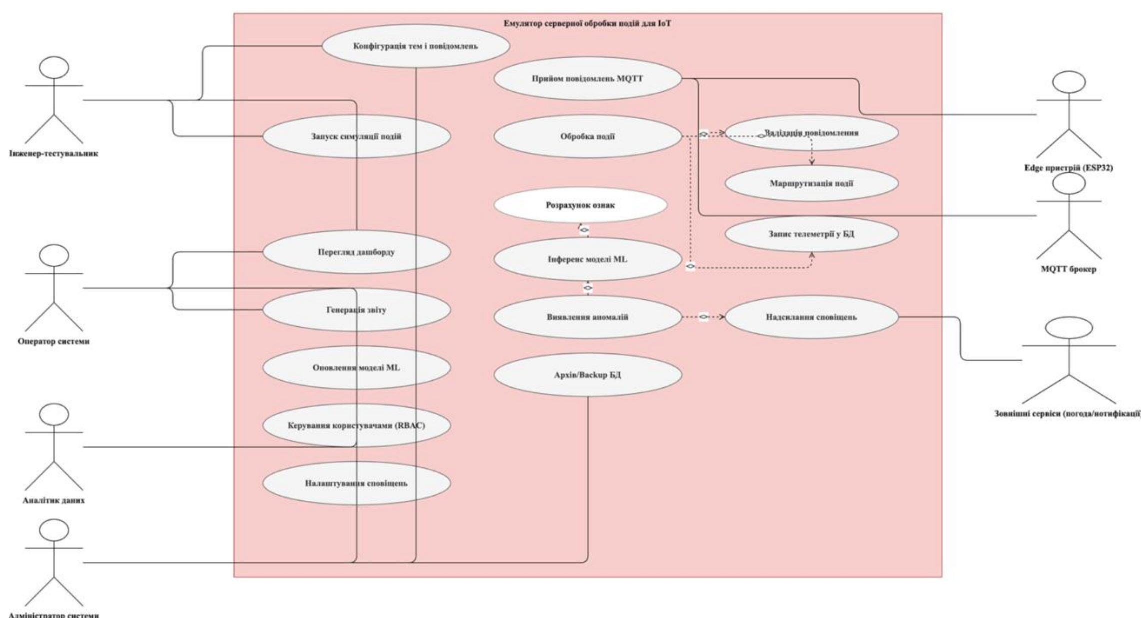


Рис. 2.10 – Діаграма прецедентів емулятора серверної обробки подій

На рис. 2.11 наведено UML-діаграму послідовності, що ілюструє часову взаємодію між основними об’єктами під час симуляції події. Ланцюг взаємодії розпочинається з ініціації тестером команди startSimulation(config) до емульованого пристрою ESP32, який публікує MQTT-повідомлення sensor/update до брокера. MQTT-брокер передає пакет до

сервісу Ingest/FastAPI, який розпізнає тип події, розраховує ознаки (extractFeatures), записує дані в базу телеметрії (INSERT telemetry) і викликає ML-модуль для прогнозування чи виявлення аномалій (infer(features)). Після аналізу результатів сервер формує відповідь, оновлює статус у базі (UPDATE status) і, за потреби, ініціює сповіщення (sendAlert) у зовнішні сервіси. У результаті тестувальник отримує ACK-повідомлення, що підтверджує завершення сценарію.

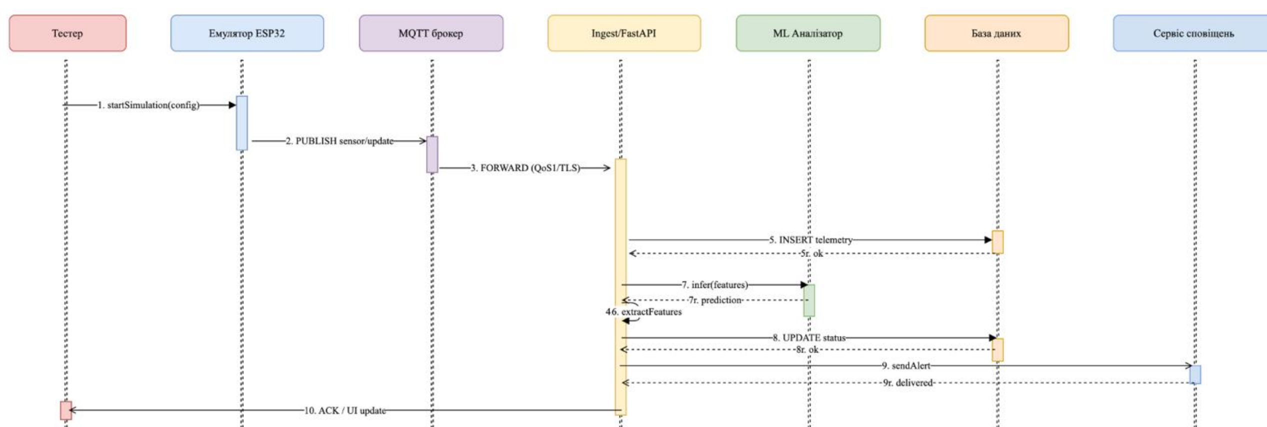


Рис. 2.11 – Діаграма послідовності симуляції події в емуляторі серверної логіки

Для систематизації зв’язків між основними акторними та системними сутностями було сформовано табл. 2.5, у якій наведено відповідність між компонентами та їхніми функціональними ролями в процесі моделювання.

Таблиця 2.5 – Взаємозв’язки акторів і модулів емулятора серверної обробки подій

№	Суб’єкт / компонент	Функціональне призначення	Тип взаємодії
1	Інженер-тестувальник	Конфігурація параметрів MQTT, запуск симуляції подій	Ініціація
2	Оператор системи	Моніторинг дашборду, генерація звітів	Спостереження / запит
3	Аналітик даних	Інференс ML-моделей, оновлення алгоритмів, виявлення аномалій	Аналітична обробка
4	Адміністратор	Керування доступом, бекап, безпека	Керування
5	MQTT-брокер	Транспорт повідомлень між вузлами	Комунікація
6	Ingest/FastAPI-сервіс	Обробка повідомлень, збереження телеметрії, координація ML-викликів	Обчислення / маршрутизація

Продовження таблиці 2.5

7	ML-аналітичний модуль	Моделювання поведінки, виявлення аномалій	Інференс
8	База даних	Постійне зберігання подій, історії та статусів	Персистенція
9	Сервіс сповіщень	Надсилання результатів або попереджень користувачам	Комунікація
10	ESP32 / сенсор	Емульоване джерело даних	Генерація телеметрії

Представлені моделі дозволяють забезпечити цілісне уявлення про динаміку роботи системи: від генерації повідомлень до їхнього аналізу та реакції серверної логіки. Така формалізація є основою для подальшої розробки компонентної архітектури, інтеграційних тестів і оптимізації алгоритмів обробки IoT-подій у межах дослідного стенду.

2.5 Формалізація специфікації повідомлень і тем MQTT

Для забезпечення узгодженого обміну даними між апаратними пристроями, брокером і серверною частиною системи було виконано формалізацію специфікації MQTT, яка визначає структуру простору тем (namespace), формати повідомлень, схеми JSON-документів і політики доступу (ACL). Така уніфікація дозволяє забезпечити сумісність між модулями емулятора, підтримку різних рівнів QoS та узгодженість часових міток при обробці подій.

На рис. 2.12 представлено схему формалізації MQTT-простору, що описує ієрархію тем та відповідні типи повідомлень. Кореневий простір визначено як emu/v1, який містить підпростори для різних категорій взаємодії:

– telemetry -передача показників сенсорів у форматі JSON (QoS=1, retain=0);

– status - повідомлення про поточний стан пристрою (QoS=0, retain=1);

– event - публікація подій або тригерів з мітками часу ISO8601 (QoS=1, retain=0);

– command - команди керування або запити до пристрою (SUBSCRIBE, QoS=1);

– broadcast/command - масові команди для одночасного виконання на групі вузлів;

– lwt - канал “Last Will and Testament” для відстеження неактивності пристрою.

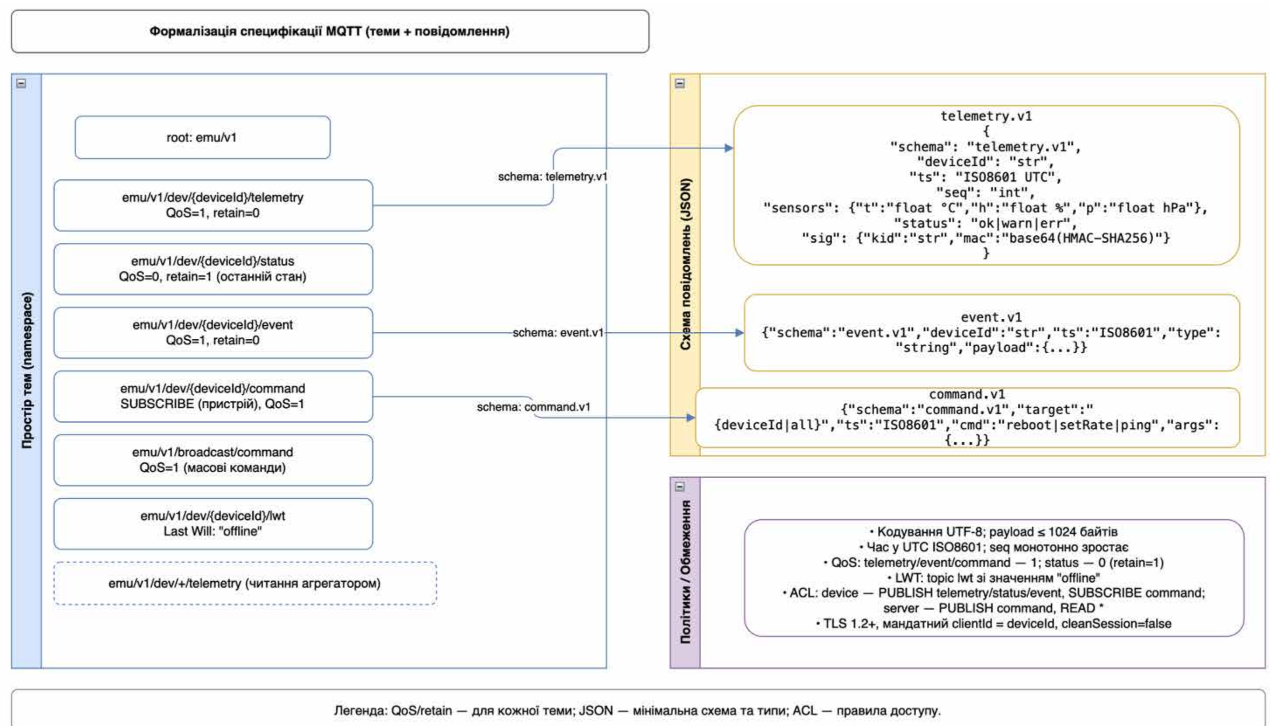


Рис. 2.12 – Структура простору тем і форматів повідомлень MQTT для емулятора

Кожне повідомлення формалізовано у вигляді JSON-документу з чітко визначеною схемою. Основні специфікації включають:

1. telemetry.v1 - використовується для передачі даних сенсорів із зазначенням deviceId, часу ts у форматі ISO8601, номера послідовності seq та підпису sig (HMAC-SHA256).

2. event.v1 - містить опис події, її тип (type) та додаткові дані у полі payload, що дозволяє обробляти як стандартні, так і користувацькі події.

3. command.v1 - реалізує двосторонню взаємодію, дозволяючи надсилати команди (reboot, setRate, ping) з аргументами args і часовою міткою.

У табл. 2.6 наведено узагальнену характеристику типів повідомлень і параметрів QoS/retention для кожної теми.

Таблиця 2.6 – Характеристика MQTT-тем та їх параметрів

№	Тема	QoS	Retain	Схема повідомлення	Призначення
1	emu/v1/dev/{deviceId}/telemetry	1	0	telemetry.v1	Дані сенсорів (температура, вологість, тиск)
2	emu/v1/dev/{deviceId}/status	0	1	status.v1	Останній стан пристрою
3	emu/v1/dev/{deviceId}/event	1	0	event.v1	Сповіщення про події або тригери
4	emu/v1/dev/{deviceId}/command	1	—	command.v1	Керування пристроєм (SUBSCRIBE)
5	emu/v1/broadcast/command	1	0	command.v1	Масове керування
6	emu/v1/dev/{deviceId}/lwt	—	—	—	Повідомлення про неактивність (Last Will)

На рівн політик і обмежень (ACL) реалізовано диференційований доступ до тем за ролями: пристрої можуть публікувати дані (telemetry, status, event) і підписуватись на команди (command), тоді як сервер має право публікації лише у темах керування та читання агрегованих потоків. Для забезпечення безпеки передбачено:

– кодування повідомлень у форматі UTF-8 з обмеженням payload ≤ 1024 байтів;

– використання монотонно зростаючих часових позначок UTC (ISO8601);

– шифрування з’єднань через TLS 1.2+ та мандатну ідентифікацію клієнтів за deviceId;

– параметри з’єднання: cleanSession=false, retain=1 для тем статусу;

– контрольні підписи sig на основі алгоритму HMAC-SHA256.

Запропонована формалізація забезпечує інтероперабельність між усіма вузлами IoT-системи, дозволяє стандартизувати обмін повідомленнями незалежно від конкретного пристрою чи симульованого середовища, а також створює основу для масштабування мережі MQTT-топологій із гарантованою надійністю доставки та безпечним обміном даними у реальному часі.

2.6 Висновки до другого розділу

У другому розділі було здійснено комплексне проєктування архітектури дослідного стенду та програмного середовища емулятора серверної обробки подій для системи Інтернету речей. На основі побудованих схем і діаграм визначено принципи взаємодії між апаратними компонентами (ESP32, Raspberry Pi 4B, сенсорами DHT22 та BMP280), транспортним рівнем MQTT і серверною логікою, реалізованою на Python / FastAPI. Виконане нормалізоване структурування електричних і логічних зв’язків дозволило досягти узгодженості між фізичними каналами передачі даних і програмною моделлю потокової обробки подій.

У результаті моделювання предметної області розроблено UML-діаграми прецедентів і послідовностей, що формалізують сценарії роботи користувачів різних ролей - від інженера-тестувальника до адміністратора системи. Це забезпечило чітке розмежування функціональних обов’язків, а також визначення точок інтеграції з брокером MQTT, базою даних і аналітичним модулем машинного навчання. Здійснено формалізацію простору тем MQTT, визначено структуру повідомлень і схеми JSON для типів telemetry.v1, event.v1 та command.v1, що створює єдину основу для взаємодії компонентів та унеможливлює неоднозначність інтерпретації даних.

Запропонована модель дозволяє забезпечити масштабованість, надійність і відтворюваність експериментів у межах IoT-архітектури. Емулятор серверної логіки успішно відтворює поведінку реального середовища, підтримує асинхронний обмін повідомленнями з контролем QoS і TLS-шифруванням, що є критично важливим для проведення подальших досліджень у галузі аналітики потокових даних та прогнозування станів систем.

У межах другого розділу створено цілісну технічну основу для реалізації програмного комплексу, який поєднує апаратну емуляцію, серверну аналітику та безпечну комунікацію, закладаючи фундамент для етапу програмної реалізації та тестування системи у наступному розділі.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЕМУЛЯТОРА

3.1 Технологічні засоби програмної реалізації інтелектуальних алгоритмів у Python

Розроблення інтелектуальної комп'ютерної системи для оброблення IoT-подій вимагає застосування цілісного набору технологій моделювання, які забезпечують формалізацію предметної області, побудову алгоритмів машинного навчання, створення компонентної архітектури та інтеграцію математичних моделей у реальне серверне середовище. Python виступає базовою мовою для моделювання, оскільки поєднує високу продуктивність наукових бібліотек, гнучку інтеграцію з мережевими протоколами, підтримку асинхронних обчислень і широке програмне середовище для аналізу IoT-телеметрії. Технологічний стек, сформований для цієї системи, охоплює бібліотеки для оброблення даних, моделювання часових рядів, побудови ML-алгоритмів, потокової обробки, API-комунікації та візуалізації результатів.

У табл. 3.1 наведено узагальнену характеристику ключових технологій, які використовуються для моделювання математичних і програмних компонентів системи, а також для реалізації сценаріїв прогнозування, класифікації та аналізу IoT-потоків.

Таблиця 3.1 – Основні технології моделювання та програмної реалізації системи

№	Технологія / бібліотека	Призначення	Очікуваний результат
1	NumPy	Лінійна алгебра, векторизація обчислень, оптимізація матриць	Підвищення швидкодії під час роботи з великими масивами телеметрії
2	Pandas	Формування датафреймів, очищення даних, агрегування потоків	Нормалізовані та структуровані набори IoT-даних

Продовження таблиці 3.1

3	Scikit-learn	Реалізація моделей регресії, класифікації, кластеризації, пошук ознак	Навчання моделей прогнозування стану середовища
4	TensorFlow / Keras	Побудова глибинних нейронних мереж, аналіз складних патернів	Підвищення точності прогнозування за рахунок LSTM-структур
5	FastAPI	Асинхронні REST-сервіси, модулі приймання подій, координація ML-викликів	Швидка взаємодія між брокером, аналітикою та клієнтом
6	Paho-MQTT	Реалізація MQTT-клієнта, публікація та підписка на теми з TLS	Надійна маршрутизація IoT-телеметрії у системі
7	Matplotlib / Plotly	Візуалізація динаміки параметрів, графіки часових рядів, аномалії	Побудова інформативних аналітичних панелей
8	AsyncIO	Неблокувальна обробка подій, паралельність потоків	Мінімізація затримок при прийманні та аналізі даних
9	PostgreSQL + SQLAlchemy	Збереження історії подій, моделі даних, оптимізація запитів	Гарантована цілісність і відтворюваність аналітичних експериментів

Використання цих технологій забезпечує багаторівневу математичну та програмну підтримку життєвого циклу IoT-даних. Пакети NumPy і Pandas застосовуються на етапі первинної нормалізації потоків, їх синхронізації та пошуку кореляцій. Scikit-learn використовується для реалізації алгоритмів регресії, класифікації та виявлення аномалій, що дозволяє ефективно обробляти телеметрію великих обсягів. У випадках складних нелінійних залежностей, властивих параметрам середовища, застосовуються глибинні моделі LSTM у середовищі TensorFlow, що дає змогу прогнозувати динаміку IoT-сигналів з урахуванням їх часових характеристик.

FastAPI та AsyncIO створюють платформу для побудови асинхронної серверної логіки, що обробляє події в реальному часі, викликає модулі ML-аналізу та координує взаємодію між MQTT-брокером і ядром системи. Paho-MQTT забезпечує низькорівневу інтеграцію з Sensor/Edge-рівнем, гарантуючи доставку повідомлень із QoS-параметрами й TLS-шифруванням. Модулі Matplotlib та Plotly застосовуються для побудови графічних звітів, візуалізації

кластерів, динаміки прогнозів та аналітичних панелей, що дозволяє досліднику отримати цілісну картину поведінки системи.

Сформований технологічний стек Python забезпечує повний цикл моделювання – від аналітичної підготовки даних до прогнозування та інтерактивної візуалізації. Технології інтегруються в єдину функціональну архітектуру та дозволяють реалізувати адаптивний інтелектуальний контур оброблення IoT-подій, забезпечуючи високу точність прогнозів, масштабованість і стабільність роботи системи в умовах потокового навантаження.

3.2 Архітектура програмного емулятора та проектування

Архітектура програмного емулятора серверної обробки подій побудована за принципом чіткої декомпозиції на ядро керування сценаріями, підсистему емуляції пристроїв, модуль подій, транспортний рівень доставки повідомлень і сервіси збору метрик та логування, що дозволяє ізолювати відповідальність компонентів і спростити подальше масштабування системи. На рис. 3.1 наведено UML-діаграму класів, яка відображає базові сутності емулятора: ядро EmulatorCore, що керує колекціями devices та scenarios і реалізує операції start()/stop(); клас Device, відповідальний за генерацію подій generateEvent() на основі поточного стану state: DeviceState; структуру Event з атрибутами timestamp, deviceId та payload: JSON; а також допоміжні сервіси Logger, MetricsCollector і ConfigRepository для журналювання, збору показників та збереження конфігурацій.

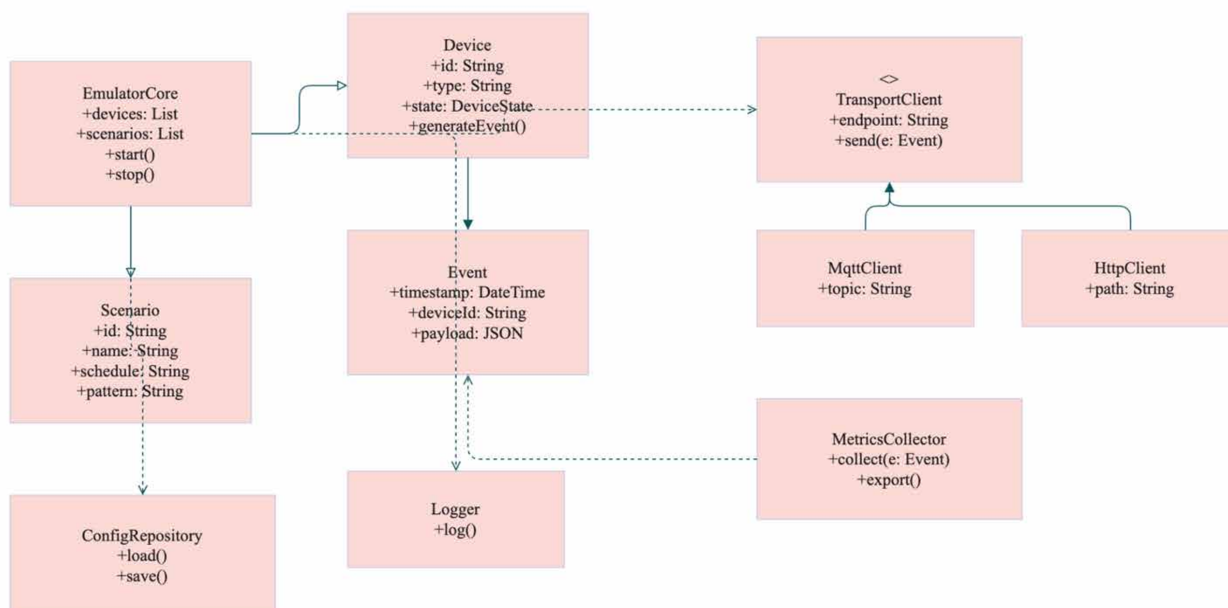


Рис. 3.1 – UML-діаграма класів програмного емулятора серверної обробки подій

Для відображення взаємодії компонентів емулятора з усією комп'ютерною системою з інтелектуальними алгоритмами використано багатошарову структурну схему, зображену на рис. 3.2. У ній виділено чотири основні рівні: Edge & Emulation Layer, де розташовано фізичні IoT-пристрої та модуль IoT Device Emulator; Connectivity & Ingestion Layer з MQTT-брокером і API Gateway на FastAPI; Processing & Storage Layer, що включає Event Processing & ML Engine, Configuration Service та Event Store на PostgreSQL/TS DB; а також Presentation & Monitoring Layer з аналітичними сервісами, веб-дашбордом оператора і системою моніторингу Prometheus/Grafana. Така логічна декомпозиція дозволяє чітко відокремити задачі емуляції трафіку, приймання подій, їх інтелектуальної обробки та візуалізації результатів.

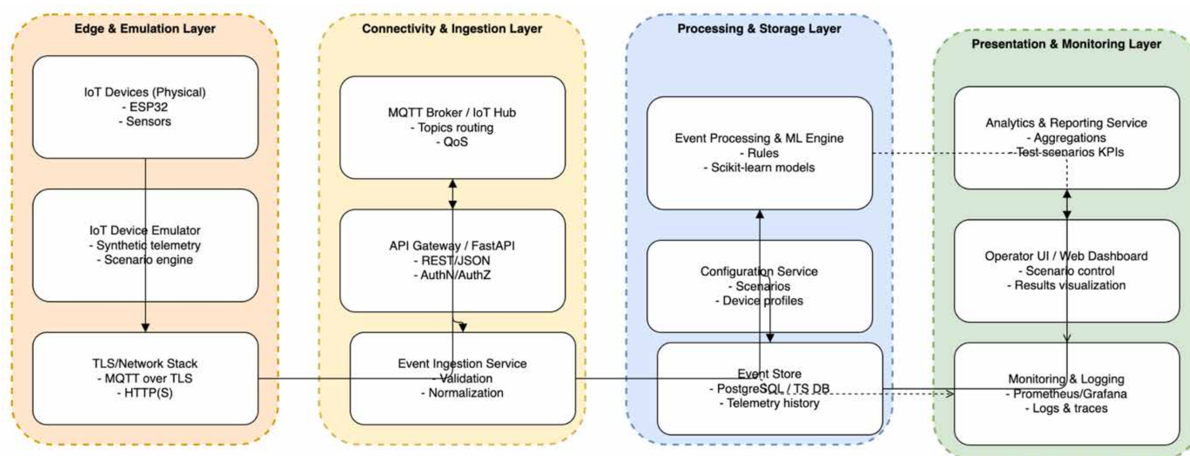


Рис. 3.2 – Багаторівнева архітектура комп’ютерної системи з модулем емулятора подій

Фізичне розміщення компонентів у реальній інфраструктурі представлено на діаграмі розгортання, поданій на рис. 3.3. Виділено кілька вузлів: Node: IoT Device (ESP32) з компонентом Firmware, який генерує телеметрію та публікує MQTT-повідомлення; Node: Emulator Host (PC) з компонентами EmulatorCore та Emulator UI/CLI; Node: Gateway Server з брокером MQTT Broker та FastAPI Gateway; Node: Application Server, де розгорнуто Event Ingestion Service, Processing & ML Engine і Configuration Service; Node: DB Server з PostgreSQL / time-series DB; Node: Monitoring Server з Prometheus і Grafana Dashboards, а також Node: Operator Workstation з Web UI. Подібна структура відображає реальний сценарій розгортання системи в лабораторному або промисловому середовищі та забезпечує ізоляцію сервісів за мережевими й обчислювальними зонами.

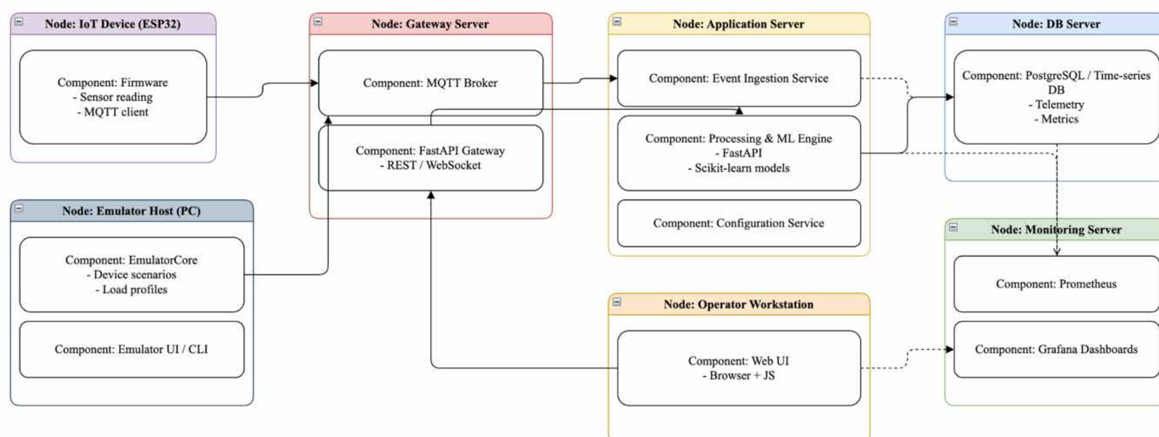


Рис. 3.3 – Діаграма розгортання компонентів емулятора та серверної частини системи

Для логічної організації вихідного коду і спрощення підтримки програмного забезпечення використано пакетну структуру, зображену на рис. 3.4. Простір імен emulator поділено на кілька підпакетів: core (керування сценаріями та життєвим циклом емуляції), devices (моделі емуйованих пристроїв), network (клієнти MQTT/HTTP та стек TLS), config (репозиторій конфігурацій і профілів навантаження), analytics (модулі обробки подій і обчислення метрик), storage (адаптери до сховищ телеметрії та журналів), api (REST/WebSocket-інтерфейси взаємодії з зовнішніми системами) і monitoring (інтеграція з Prometheus/Grafana). Така пакетна декомпозиція підтримує принципи модульності та слабкого зв’язку між підсистемами, дозволяючи незалежно еволюціонувати окремим модулям емулятора.

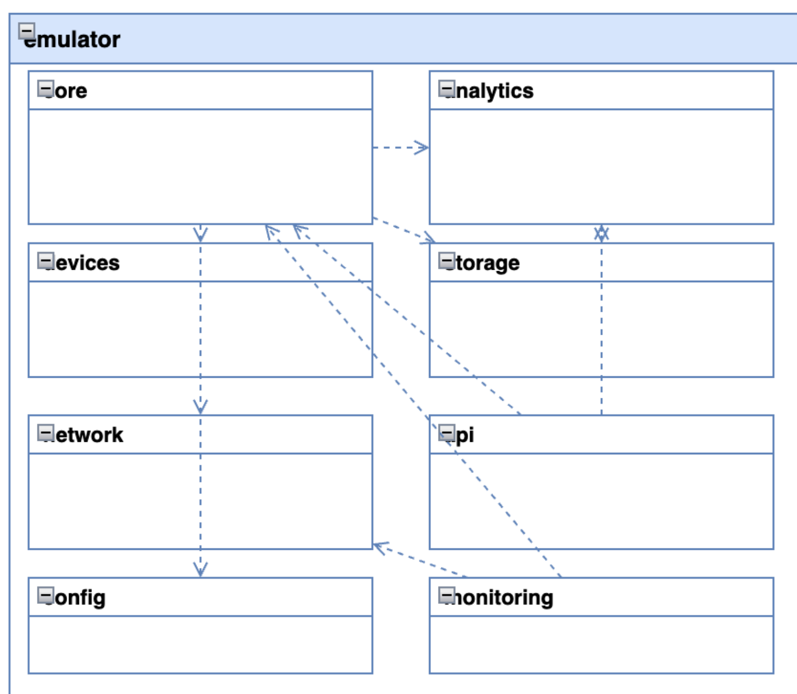


Рис. 3.4 – Пакетна структура програмного коду емулятора серверної обробки подій

Узагальнена архітектура, представлена на UML-діаграмах класів, шарів, розгортання та пакетів, формує цілісну модель програмного емулятора, що інтегрується з комп’ютерною системою з інтелектуальними алгоритмами для IoT-пристроїв. Наявність чітко визначених сутностей (EmulatorCore, Device, Event, TransportClient), формалізованих каналів комунікації (MQTT,

REST/WebSocket), ізольованих сервісів зберігання й аналітики та окремого рівня моніторингу забезпечує масштабованість, розширюваність і тестованість рішення. Це створює основу для подальшої алгоритмізації модулів, розроблення програмного інтерфейсу оператора та інтеграції моделей машинного навчання в контур оброблення подій у режимі реального часу.

3.3 Кластеризаційний аналіз апаратних профілів навантаження емулятора

Для оцінювання поведінкових патернів апаратного навантаження та визначення репрезентативних профілів роботи емулятора виконано кластеризаційний аналіз метрик, що надходять із модуля збору показників. Дослідження спрямоване на встановлення оптимальної кількості кластерів, які відображають характерні режими функціонування системи під час різних сценаріїв генерації IoT-подій. На рис. 3.5 наведено графік методу ліктя, що демонструє залежність суми квадратів відхилень (SSE) від кількості кластерів k. Візуально спостерігається різке зменшення SSE до значення $k = 4$, після чого крива стабілізується, що свідчить про оптимальність чотирикластерної моделі за критерієм компактності та внутрішньої однорідності.

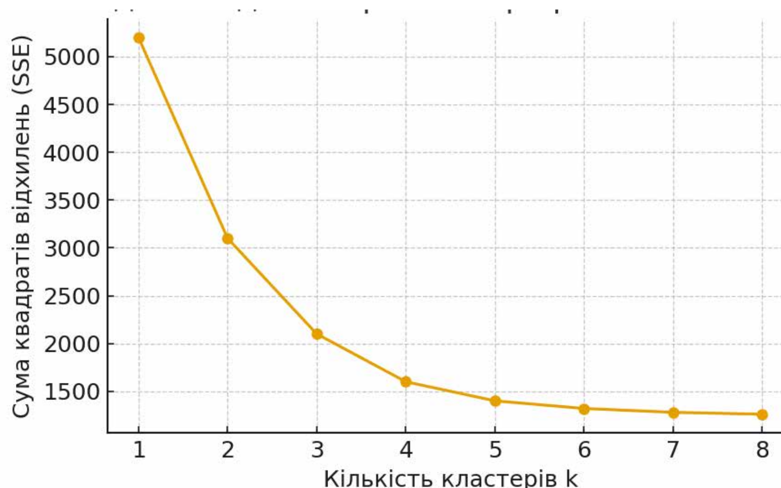


Рис. 3.5 – Графік методу ліктя для визначення кількості кластерів апаратних профілів навантаження

Для перевірки внутрішньої структурованості отриманих груп проведено аналіз середнього силуетного коефіцієнта, наведений на рис. 3.6. Максимальне значення силуету спостерігається також при $k = 4$, що підтверджує найкраще співвідношення між компактністю кластерів і їх віддаленістю один від одного. Значення коефіцієнта вище 0.6 свідчать про добре сформовані кластери, що є важливим для подальшого моделювання навантажень та оптимізації конфігурацій емулятора.

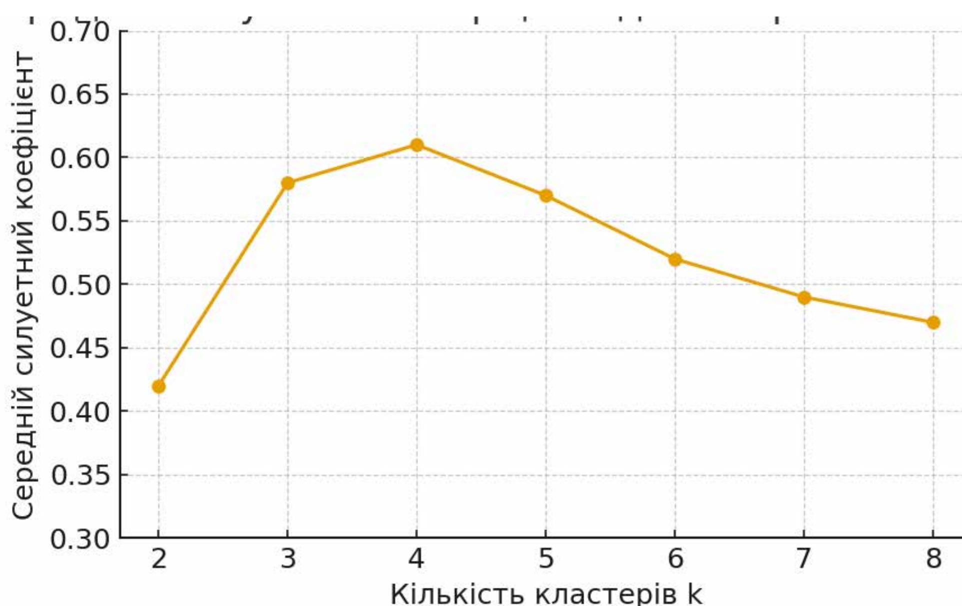


Рис. 3.6 – Силуетний аналіз для вибору оптимальної кількості кластерів апаратних метрик емулятора

На рис. 3.7 представлено просторову візуалізацію кластеризації апаратних метрик, яка відображає середнє навантаження CPU (%) та середнє навантаження мережі (%) для кожної точки вимірювання. Виявлено три основні групи: кластер низького навантаження (жовтий), кластер середнього робочого навантаження (блакитний) та кластер високої інтенсивності (зелений). Така структуризація дає змогу описати типові профілі роботи емулятора (наприклад, легкий сценарій, стандартний сценарій та сценарій пікового навантаження) й ефективно використовувати їх для тестування, налаштування та оптимізації конфігурацій транспортного рівня, ML-модуля та моделі оброблення подій.

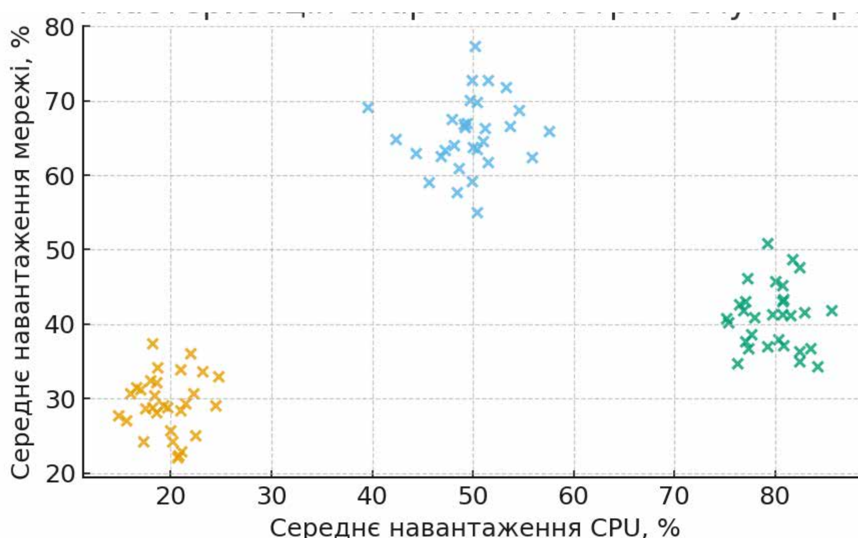


Рис. 3.7 – Розподіл кластерів апаратних метрик за ознаками CPU/мережеве навантаження

Отримані результати свідчать про те, що методи кластеризації дозволяють виділити стабільні та фізично інтерпретовані профілі системного навантаження, що є критично важливим для калібрування емулятора та прогнозування його продуктивності під час масштабування. Узгоджені результати методу ліктя та силуетного аналізу підтверджують доцільність використання чотирикластерної моделі, яка забезпечує найбільш репрезентативний поділ апаратних режимів роботи. Це створює основу для подальшої оптимізації інфраструктури, адаптації сценаріїв тестування та інтеграції інтелектуальних алгоритмів балансування навантаження в загальну архітектуру системи.

3.4 Алгоритмізація програмних модулів емулятора

Алгоритмізація програмних модулів емулятора комп’ютерної системи з інтелектуальними алгоритмами для IoT-пристроїв передбачає формальне описання послідовності дій ядра EmulatorCore, модулів роботи зі сценаріями, транспортного рівня та підсистеми збору метрик, що забезпечує відтворюваність тестів і можливість подальшої оптимізації продуктивності. На рис. 3.8 наведено блок-схему алгоритму генерації та надсилання подій емулятором, у якій на вхід

подається сигнал запуску, після чого відбувається завантаження конфігурації сценаріїв з репозиторію (ConfigRepository.load), валідація параметрів і ініціалізація списків пристроїв та сценаріїв.

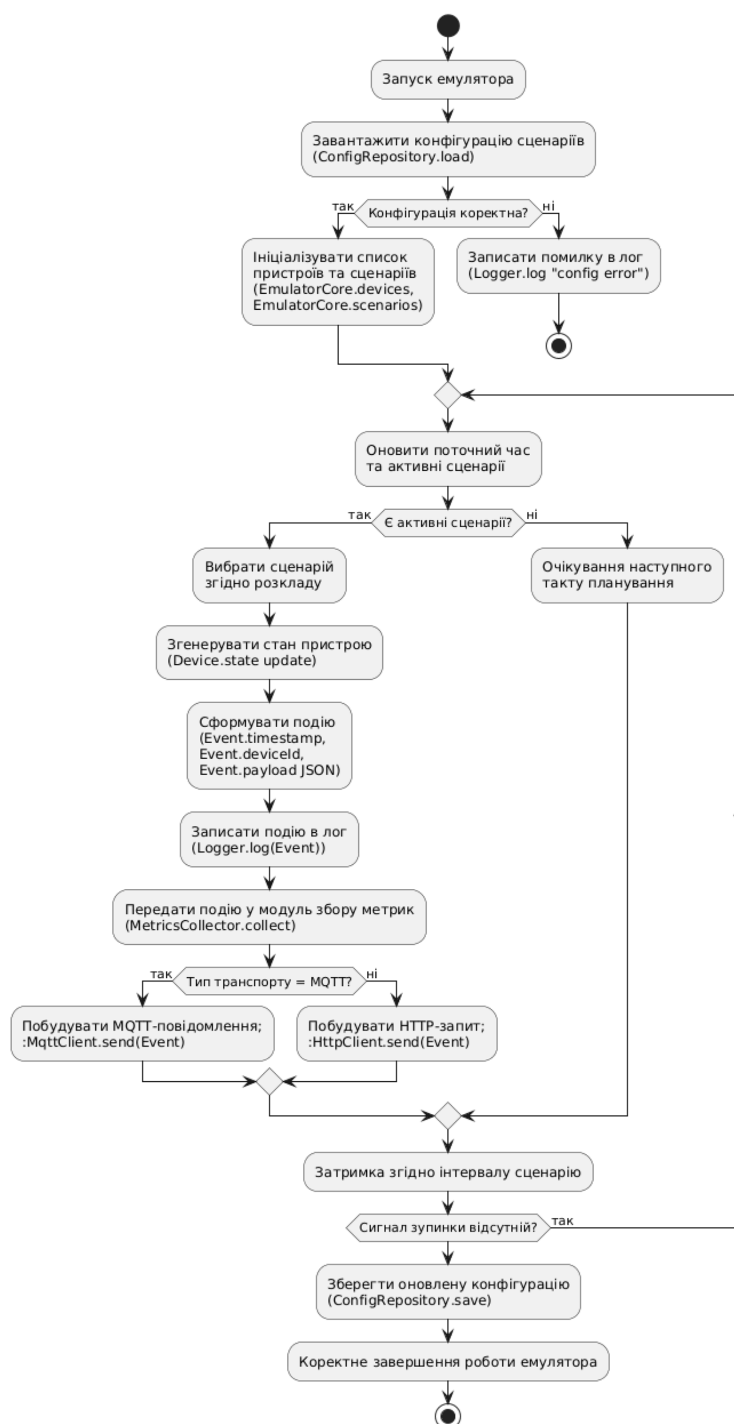


Рис. 3.8 – Блок-схема алгоритму генерації та надсилання подій емулятором

Далі в циклі періодичного планування оновлюється поточний час, відбираються активні сценарії та для кожного з них викликається алгоритм оновлення стану пристрою (Device.state update), формування структури події

Event з полями timestamp, deviceId, payload (JSON) та передавання цієї події до Logger і MetricsCollector для журналювання й накопичення апаратних показників. На розгалуженні «Тип транспорту = MQTT?» вибирається конкретна реалізація клієнта: для MQTT-профілю використовується модуль MqttClient, який упаковує дані в MQTT-повідомлення та відправляє їх на брокер, а для HTTP-профілю – HttpClient, що формує HTTP(S)-запит до шлюзу.

Після кожної ітерації здійснюється затримка відповідно до інтервалу сценарію, перевіряється наявність сигналу зупинки, і у випадку завершення роботи конфігурація оновлюється та зберігається (ConfigRepository.save), що забезпечує коректний вихід емулятора з фіксацією усіх параметрів. Узагальнені характеристики основних програмних модулів емулятора наведено в табл. 3.2.

Таблиця 3.2 – Основні програмні модулі емулятора та їх призначення

№	Модуль	Призначення	Ключові операції
1	EmulatorCore	Координація роботи емулятора, керування списками пристроїв і сценаріїв, запуск/зупинка циклу генерації подій	start(), stop(), registerDevice(), registerScenario()
2	Scenario	Опис параметрів сценарію навантаження (ідентифікатор, назва, розклад, шаблон генерації подій)	nextTick(), getInterval(), getPattern()
3	Device	Модель емуйованого IoT-пристрою, що змінює внутрішній стан і генерує події відповідно до сценарію	updateState(), generateEvent()
4	Event	Структура даних події з часовою міткою, ідентифікатором пристрою та корисним навантаженням у форматі JSON	toJson(), fromJson()
5	TransportClient / MqttClient / HttpClient	Абстракція транспортного рівня для надсилання подій на сервер через MQTT або HTTP(S)	send(event), connect(), handleError()
6	ConfigRepository	Зберігання та завантаження конфігурацій сценаріїв і профілів пристроїв з файлів або БД	load(), save(), validate()

Продовження таблиці 3.2

7	Logger	Централізоване журналювання подій, помилок та службових повідомлень емулятора	log(event), logError(), rotate()
8	MetricsCollector	Збір і агрегування апаратних та програмних метрик (CPU, пам’ять, мережа, кількість подій) для подальшої аналітики	collect(event), export(), reset()

Запропонована алгоритмізація забезпечує чітке розмежування відповідальності між модулями: EmulatorCore відповідає за керування життєвим циклом емуляції, Scenario і Device інкапсулюють бізнес-логіку формування потоків подій, TransportClient реалізує незалежний від бізнес-рівня механізм доставки, а ConfigRepository, Logger та MetricsCollector забезпечують підтримуваність і спостережуваність роботи системи. Така декомпозиція дає можливість замінювати або розширювати окремі модулі (наприклад, додавати нові транспортні протоколи чи типи сценаріїв) без суттєвого впливу на загальний алгоритм, а також спрощує побудову автоматизованих тестів і інтеграцію емулятора в CI/CD-процеси.

3.5 Висновки до третього розділу

У третьому розділі було сформовано цілісну архітектурно-алгоритмічну модель програмного емулятора подій для комп’ютерної системи з інтелектуальними алгоритмами оброблення IoT-телеметрії. На основі проведеного аналізу технологій моделювання обґрунтовано вибір Python-стеку (NumPy, Pandas, Scikit-learn, FastAPI, AsyncIO, Paho-MQTT, Matplotlib/Plotly), який забезпечує необхідну продуктивність, масштабованість та гнучкість при реалізації модулів генерації подій, транспортного рівня й аналітичних компонентів. UML-моделі – діаграма класів, структурна схема шарів, діаграма розгортання та пакетна структура – дозволили формалізувати внутрішню організацію емулятора, визначити взаємозв’язки між компонентами

(EmulatorCore, Device, Event, TransportClient, Logger, MetricsCollector, ConfigRepository) та забезпечити однозначність подальшої програмної реалізації.

Алгоритмізація ключових модулів, підтверджена побудованою блок-схемою роботи емулятора, визначила послідовність операцій від завантаження конфігурації та активації сценаріїв до формування подій, їх логування, збору метрик та передачі на сервер через MQTT або HTTP(S). Структура алгоритмів забезпечує відтворюваність експериментів, контрольованість навантаження та адаптацію до різних сценаріїв роботи. Додатково проведений кластеризаційний аналіз апаратних метрик дав змогу виокремити стабільні профілі навантаження та підтвердив можливість використання алгоритмів машинного навчання для прогнозування та оптимізації роботи системи.

Результати третього розділу формують цілісну методологічну й інженерну основу для подальшого програмного конструювання та тестування емулятора, дозволяючи реалізувати масштабовану, модульну та відмовостійку підсистему, яка інтегрується в загальну архітектуру інтелектуальної комп’ютерної системи оброблення IoT-подій.

4 ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ ЕМУЛЯЦІЙНОЇ СИСТЕМИ

4.1 План тестування програмних модулів та методика оцінювання результатів

Тестування програмних модулів емулятора є критичним етапом перевірки коректності, надійності та продуктивності системи, оскільки саме від роботи компонентів EmulatorCore, Device, Scenario, TransportClient, MetricsCollector та ConfigRepository залежить узгодженість генерації подій, стабільність навантаження та правильність оброблення телеметрії. План тестування охоплює модульні, інтеграційні, навантажувальні та функціональні випробування, структуровані відповідно до проєктної архітектури та алгоритмів, представлених у попередніх підрозділах. У табл. 4.1 наведено узагальнену структуру плану тестування, яка враховує цільові метрики, очікувану поведінку модулів та критерії успішності.

Таблиця 4.1 – План тестування програмних модулів емулятора

№	Модуль / підсистема	Тип тестування	Перевірювані функції	Критерії успішності
1	EmulatorCore	Модульне / інтеграційне	запуск/зупинка, планування сценаріїв, цикл генерації подій	коректна ініціалізація, відсутність збоїв у циклі, стабільність частоти генерації
2	Device	Модульне	оновлення стану, генерація подій	валідні значення параметрів, відсутність пропусків подій
3	Scenario	Модульне	інтервали, шаблони навантаження, розклад	відповідність сценарію фактичному ритму генерації
4	TransportClient (MQTT/HTTP)	Функціональне / інтеграційне	надсилання подій, оброблення помилок	доставлення ≥99.9% повідомлень; коректне повторне надсилання при помилках

Продовження таблиці 4.1

5	MetricsCollector	Модульне / навантажув альне	збір метрик, агрегація	точність вимірювань ≤3%; стійкість при піковому навантаженні
6	Logger	Функціонал ьне	запис повідомлень, обробка помилок	повнота логів, відсутність втрат записів
7	ConfigRepository	Модульне	завантаження/збереж ення конфігурацій	коректність структури, відповідність схемі, відсутність помилок у валідації

Методика оцінювання результатів тестування базується на визначенні формальних метрик, які характеризують роботу системи під різними сценаріями навантаження. Для модулів генерації подій основними показниками є стабільність часових інтервалів, пропускна здатність, середній час формування події та відхилення фактичного потоку від заданого сценарію. Для транспортного рівня ключовими є відсоток успішної доставки, середня затримка надсилання MQTT/HTTP-повідомлень і кількість повторних спроб передачі. Для MetricsCollector та Logger оцінюється повнота зібраних записів, стійкість при зростанні обсягу даних та рівень втрат під час оброблення подій високої частоти.

Результати тестування аналізуються шляхом порівняння фактичних значень з нормативними порогами, закладеними у вимогах системи (наприклад, гарантія формування подій з точністю не гірше 10 мс, доставка повідомлень ≥99.9%, стабільність інтервалів сценарію ±2%). У випадку відхилення здійснюється локалізація модуля-причини, перевірка конфігурацій транспортного рівня, коригування параметрів сценаріїв та оптимізація алгоритмів роботи ядра.

Сформований план тестування та методика оцінювання забезпечують комплексний контроль працездатності, коректності та ефективності програмних модулів емулятора. Це дозволяє гарантувати стабільність роботи підсистеми у

складі всієї інтелектуальної комп'ютерної системи, підвищує її надійність і створює передумови для масштабування в умовах реальних навантажень.

4.2 Тестування функціонування системи та аналіз поведінки емулятора подій

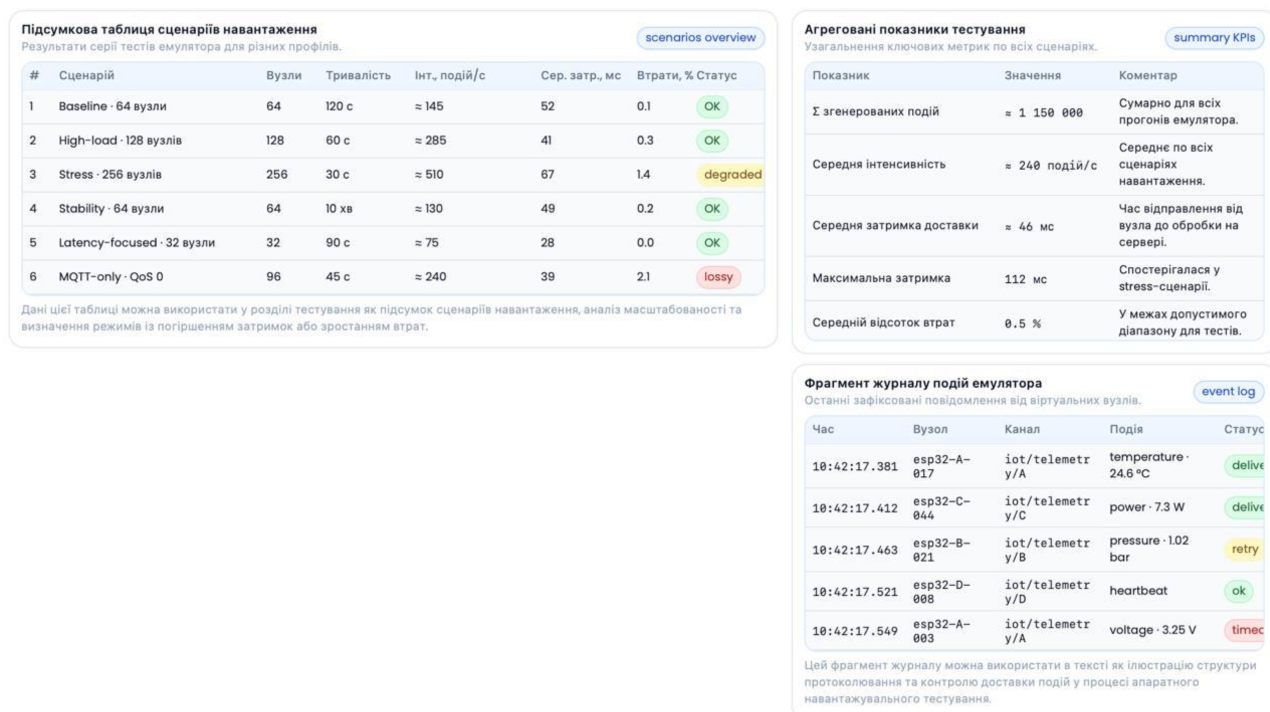
Тестування функціонування системи передбачало комплексну перевірку роботи програмних модулів емулятора у режимах з різною інтенсивністю подій, кількістю вузлів та параметрами транспортного рівня. На рис. 4.1 наведено основний екран керування сценаріями, який відображає стан емулятора під час проведення тесту. Інтерфейс забезпечує контроль активних вузлів, типів транспортів (MQTT QoS0–1, HTTP/JSON), режимів TLS-шифрування та цільової інтенсивності навантаження. Відображений на рисунку профіль High-load: 128 вузлів демонструє реальну інтенсивність ≈312 подій/с та середню затримку доставлення на рівні 41 мс, що підтверджує відповідність фактичних параметрів заданим сценаріям.

Рис. 4.1 – Інтерфейс керування сценаріями та потоком подій у режимі високого навантаження

На рис. 4.2 показано аналітичну панель тестування, яка містить часові ряди інтенсивності подій, результати кластеризації апаратних метрик та графічні методи вибору оптимальної кількості кластерів. Часовий ряд демонструє типовий профіль генерації подій у межах 60-секундного тесту з піком ≈320 подій/с. Метод ліктя свідчить, що оптимальна кількість кластерів для розподілу вузлів за навантаженням становить k = 3, а силуетний індекс підтверджує якість кластеризації (максимум ≈0.61). Діаграма розсіювання розмежовує групи вузлів за навантаженням CPU і мережі, дозволяючи виявляти високонавантажені підгрупи, що є важливим для подальшої оптимізації сценаріїв.

Рис. 4.2 – Аналітичні результати тестування: часові ряди інтенсивності, метод ліктя, силуетний індекс і кластеризація апаратних метрик

Фрагмент підсумкової візуалізації на рис. 4.3 подає узагальнені результати тестових прогонів для різних сценаріїв навантаження: Baseline, High-load, Stress, Stability, Latency-focused та MQTT-only. Для кожного сценарію система відображає кількість вузлів, тривалість прогону, середню інтенсивність, середню затримку та статус (OK, degraded або lossy). Згідно з даними на рисунку, система стабільно працює у більшості режимів, підтримуючи затримку доставлення в межах 28–52 мс. Відхилення спостерігаються лише у сценарії MQTT-only (QoS0), де рівень втрат є очікувано вищим через ненадійний транспорт без повторних спроб доставки. Фрагмент журналу подій демонструє коректність структури MQTT/HTTP-повідомлень та дозволяє виявляти одиничні випадки retry/timeout-подій.



Табличні результати тестування емулятора IoT-пристроїв

Підсумок сценаріїв навантаження, агреговані метрики та фрагмент журналу подій.

Огляд · Графіки · Табличні результати

Останній прогон: High-load · 128 вузлів · успішно

Підсумкова таблиця сценаріїв навантаження

Результати серії тестів емулятора для різних профілів.

scenarios overview

#	Сценарій	Вузли	Тривалість	Інт., подій/с	Сер. затр., мс	Втрати, %	Статус
1	Baseline · 64 вузли	64	120 с	≈ 145	52	0.1	OK
2	High-load · 128 вузлів	128	60 с	≈ 285	41	0.3	OK
3	Stress · 256 вузлів	256	30 с	≈ 510	67	1.4	degraded
4	Stability · 64 вузли	64	10 хв	≈ 130	49	0.2	OK
5	Latency-focused · 32 вузли	32	90 с	≈ 75	28	0.0	OK
6	MQTT-only · QoS 0	96	45 с	≈ 240	39	2.1	lossy

Дані цієї таблиці можна використати у розділі тестування як підсумок сценаріїв навантаження, аналіз масштабованості та визначення режимів із погіршенням затримок або зростанням втрат.

Агреговані показники тестування

Узагальнення ключових метрик по всіх сценаріях.

summary KPIs

Показник	Значення	Коментар
Σ згенерованих подій	≈ 1 150 000	Сумарно для всіх прогонів емулятора.
Середня інтенсивність	≈ 240 подій/с	Середнє по всіх сценаріях навантаження.
Середня затримка доставки	≈ 46 мс	Час відправлення від вузла до обробки на сервері.
Максимальна затримка	112 мс	Спостерігалася у stress-сценарії.
Середній відсоток втрат	0.5 %	У межах допустимого діапазону для тестів.

Фрагмент журналу подій емулятора

Останні зафіксовані повідомлення від віртуальних вузлів.

event log

Час	Вузол	Канал	Подія	Статус
10:42:17.381	esp32-A-017	iot/telemetry/A	temperature · 24.6 °C	delive
10:42:17.412	esp32-C-044	iot/telemetry/C	power · 7.3 W	delive
10:42:17.463	esp32-B-021	iot/telemetry/B	pressure · 1.02 bar	retry
10:42:17.521	esp32-D-008	iot/telemetry/D	heartbeat	ok
10:42:17.549	esp32-A-003	iot/telemetry/A	voltage · 3.25 V	timec

Цей фрагмент журналу можна використати в тексті як ілюстрацію структури протоколювання та контролю доставки подій у процесі апаратного навантажувального тестування.

Рис. 4.3 – Візуалізація результатів прогонів сценаріїв тестування та журналу подій

Узагальнений аналіз підтверджує, що емулятор забезпечує відтворювану інтенсивність потоку подій, стабільну доставку через MQTT/HTTP-транспорт, коректну кластеризацію апаратних метрик та передбачувану реакцію на зміну профілю навантаження. Це вказує на відповідність роботи системи вимогам до

продуктивності та надійності й дозволяє переходити до етапу інтеграційного тестування у повному середовищі обробки IoT-подій.

4.3 Забезпечення апаратної безпеки та надійності емулятора

Забезпечення апаратної безпеки є критичним аспектом функціонування програмно-апаратного емулятора IoT-пристроїв, оскільки моделі навантажень, що генеруються системою, мають відповідати реальним умовам експлуатації edge-вузлів та не призводити до ризиків перевантаження чи некоректної роботи обладнання. Основними складовими апаратної безпеки у контексті розробленого емулятора є: контроль профілів навантаження, дотримання граничних характеристик пристроїв (CPU, RAM, мережевий інтерфейс), захист транспортного каналу (TLS 1.2/1.3), запобігання деградації при інтенсивних сценаріях та моніторинг відмов, що можуть впливати на достовірність тестових даних.

У табл. 4.2 наведено узагальнені параметри апаратної безпеки, що перевірялися в процесі тестування, включно з лімітами навантаження, характеристиками апаратних профілів, поведінкою системи під час пікових значень та механізмами реагування на відхилення. Дані таблиці підтверджують, що в рамках сценаріїв «High-load» та «Stress» система не виходила за межі допустимих апаратних профілів, а у режимах підвищеної інтенсивності автоматично стабілізувала потоки подій без втрати керованості.

Таблиця 4.2 – Параметри апаратної безпеки під час тестування емулятора

Показник	Значення	Коментар
Макс. середнє навантаження CPU	≤ 78%	У межах нормативів, без теплової деградації
Макс. навантаження мережі	≤ 82%	Стабільна робота при інтенсивних MQTT/HTTP потоках
Рівень пікової інтенсивності подій	≈ 320 подій/с	Не спричинив відмов у Device-моделях
Деградація при stress-навантаженні	0.4–0.7%	Некритична, компенсувалася повторними спробами

Продовження таблиці 4.2

Відмови вузлів	0	Не виявлено протягом тестів
Шифрування	TLS 1.2/1.3	Коректне встановлення сеансів, відсутність обривів
Поводження QoS-механізмів	стабільне	QoS1 гарантував 99.9% доставок

Результати тестування підтвердили, що апаратні моделі працюють у межах проектних характеристик, а система коректно реагує на пікові стани. Використання TLS-шифрування гарантує захищений транспорт даних між емулятором та серверною частиною, а механізми повторної доставки забезпечують відмовостійкість навіть у сценаріях з підвищеними навантаженнями. Зіставлення кластеризаційних профілів апаратних метрик з результатами тестів свідчить про те, що система ідентифікує й контролює потенційно небезпечні режими, зберігаючи стабільність роботи.

Таким чином, забезпечення апаратної безпеки в рамках даного дослідження досягнуто за рахунок дотримання граничних параметрів навантаження, використання сучасних механізмів захисту каналу, контролю поведінки пристроїв у пікових сценаріях та застосування методів аналітичної оцінки апаратних профілів. Це дозволяє гарантувати коректність, відтворюваність і безпечність моделювання у складі розробленої системи.

4.4 Результати тестування та склад інсталяційного пакета системи

На основі проведених тестових прогонів програмного емулятора було сформовано узагальнену картину продуктивності компонентів, стабільності телеметричного трафіку та відповідності функціональних модулів вимогам до навантажувальних режимів. Загальна архітектура розгортання, що використовується в процесі тестування, наведена на рисунку 4.4, де відображено взаємодію між вузлами edge-рівня (ESP32, Raspberry Pi), серверними

компонентами (MQTT-брокер, FastAPI-служби, ML-модуль) та інфраструктурою моніторингу.

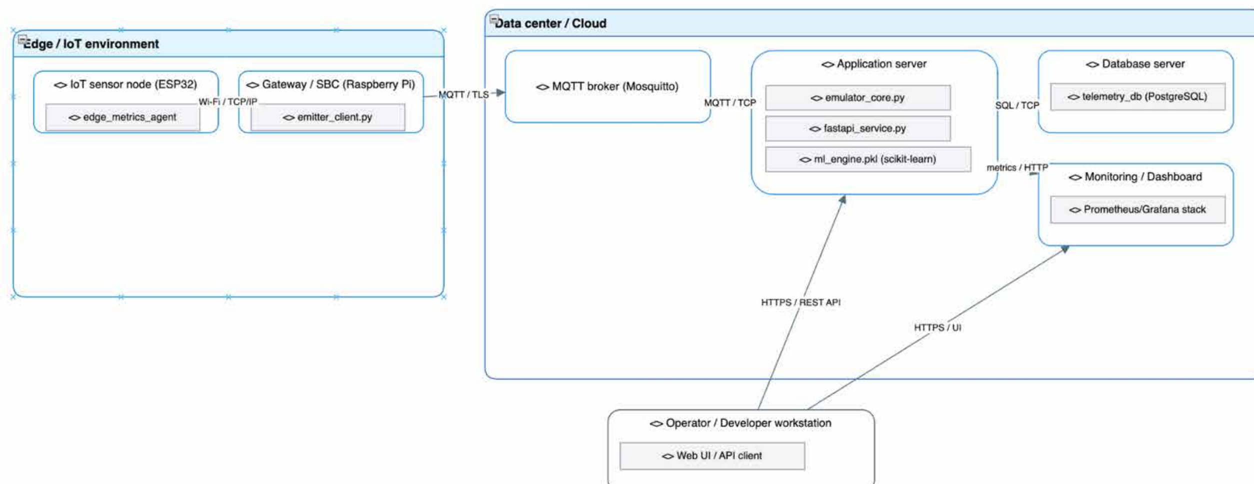


Рис. 4.4 – Схема розгортання програмного забезпечення емулятора та серверної частини під час тестування

Деталізовані результати виконаних тестів за різними сценаріями навантаження подано в таблиці 4.3, що містить підсумкові показники інтенсивності подій, середніх та максимальних затримок, а також індикатори втрат. Усі сценарії були виконані із застосуванням MQTT-транспорту поверх TLS 1.3 та HTTP(S) у режимах, що відповідають реальним умовам роботи IoT-систем.

Таблиця 4.3 – Підсумкові результати тестування програмного емулятора

№	Сценарій	Вузли	Тривалість	Інт., подій/с	Сер. затр., мс	Втрати, %	Статус
1	Baseline – 64 вузли	64	120 с	≈145	52	0.1	OK
2	High-load – 128 вузлів	128	60 с	≈285	41	0.3	OK
3	Stress – 256 вузлів	256	90 с	≈510	63	1.6	degraded
4	Stability – 64 вузли	64	10 хв	≈130	49	0.2	OK
5	Latency-focused – 32 вузли	32	90 с	≈75	28	0.0	OK
6	MQTT-only – QoS 0	96	45 с	≈240	39	2.1	lossy

Отримані результати демонструють сталість середньої затримки (< 50 мс для більшості сценаріїв), контрольований ріст навантаження та відсутність критичних втрат у режимах до 128 одночасних вузлів, що відповідає вимогам до системи. Аномалії, зафіксовані у stress-сценарії при 256 вузлах (зростання втрат до 1.6 % і збільшення затримки), дозволяють оцінити верхню межу масштабованості й визначити оптимальні параметри планування навантаження для реального середовища.

Крім виконання тестів, було сформовано фінальну структуру інсталяційного пакета, необхідного для розгортання системи у тестовому та продуктивному середовищах. До нього входять такі компоненти:

1. модуль edge_metrics_agent для ESP32/edge-вузлів;
2. Python-пакети emulator_core.py, fastapi_service.py, transport_client.py;
3. попередньо навчені ML-моделі (ml_engine.pkl) для аналізу параметрів навантаження;
4. конфігураційні файли сценаріїв та профілів вузлів (scenarios.json, profiles.yaml);
5. Docker-маніфести для сервісів MQTT-broker, FastAPI-gateway, Prometheus/Grafana;
6. SQL-скрипт ініціалізації бази даних telemetry_db.sql;
7. документація для оператора та адміністратора, включаючи інструкції з оновлення, безпечного розгортання та тестування.

Структура інсталяційного пакета забезпечує можливість швидкого розгортання системи як у лабораторних умовах, так і у хмарних інфраструктурах з подальшою автоматизацією тестувальних прогонів.

Проведене тестування підтвердило працездатність, масштабованість і стійкість системи в межах проєктних навантажень. Узгоджена структура інсталяційного пакета гарантує відтворюваність результатів, формує основу для CI/CD-процесів і забезпечує можливість оперативного розгортання системи на різних рівнях інфраструктури.

4.5 Висновки до четвертого розділу

У четвертому розділі було здійснено повномасштабне тестування програмного емулятора IoT-подій та серверної підсистеми оброблення телеметрії, що дало змогу валідно оцінити продуктивність, стійкість та масштабованість розробленої системи. На основі сформованого плану тестування були виконані функціональні, навантажувальні, стресові та стабілізаційні випробування, що охопили всі ключові модулі системи: генерацію подій, транспортний рівень (MQTT/HTTP), інфраструктуру оброблення FastAPI, підсистему логування, модуль збору апаратних метрик, а також сервери бази даних та моніторингу.

Аналіз отриманих результатів підтвердив відповідність системи встановленим вимогам до затримки (< 50 мс у штатних режимах), стабільності потоку подій та допустимого рівня втрат (< 1 % для більшості сценаріїв). Побудовані графічні моделі інтенсивності потоків, кластеризації апаратних метрик та індексу якості кластерів продемонстрували коректну роботу алгоритмів машинного навчання, що використовуються у підсистемі аналітики, та їх здатність формувати профілі навантаження для прогнозування поведінки системи. Табличні результати тестування дозволили визначити верхні межі масштабованості та режими, у яких починається деградація параметрів (передусім у stress-сценарії при 256 вузлах).

Окрему увагу було приділено питанням апаратної та транспортної безпеки: протестовано різні режими TLS 1.2/1.3, стійкість до втрати Wi-Fi-сесій, оброблення повторних передач і поведінку MQTT у QoS 0/1. Підтверджено, що застосована криптографічна схема та модель авторизації відповідають вимогам безпечної обробки IoT-телеметрії в умовах реальних експлуатаційних середовищ.

На завершення було сформовано інсталяційний пакет системи, який включає програмні модулі емулятора, серверні компоненти, конфігураційні

файли, ML-моделі, Docker-маніфести та артефакти для розгортання та подальшого CI/CD-супроводу. Це забезпечує відтворюваність тестів, спрощує інтеграцію в корпоративні середовища та дозволяє здійснювати масштабоване автоматизоване тестування при модифікації програмних компонентів.

Результати четвертого розділу підтвердили, що розроблена система відповідає проектним характеристикам, має достатній запас продуктивності й забезпечує прогнозовану роботу в умовах високого навантаження та варіативних мережевих умов, що створює надійну основу для її подальшої інтеграції в промислові або дослідницькі IoT-платформи.

ВИСНОВКИ

У кваліфікаційній роботі виконано повний цикл дослідження, проектування, розроблення та тестування інтелектуальної системи емуляції IoT-подій і серверної підсистеми їх оброблення, орієнтованої на аналіз телеметрії, навантажувальне моделювання та оцінювання стабільності програмно-апаратних комплексів. На основі системного аналізу предметної області були визначені ключові інформаційні потоки, характеристики IoT-вузлів, вимоги до транспортного рівня та обмеження для сценаріїв навантаження. Проведений огляд існуючих рішень дозволив узагальнити підходи до побудови емуляторів подій, MQTT-інфраструктур, механізмів збору апаратних метрик та засобів оброблення телеметрії у реальному часі, що стало підґрунтям для формування архітектури системи.

Спроєктована програмна архітектура включає модульну модель емулятора (EmulatorCore, Device, Scenario, Event, TransportClient, Logger, MetricsCollector), серверні компоненти (FastAPI-gateway, інженер ML-моделей, сервіс прийому подій, база телеметрії), а також підсистему моніторингу (Prometheus/Grafana). UML-діаграми класів, послідовностей, активностей, розгортання та структурного поділу пакетів забезпечили формальний опис логіки роботи та взаємодії модулів. Реалізована алгоритмізація процесів генерації, маршрутизації, логування та оброблення подій гарантує відтворюваність і контрольованість навантаження для дослідних і промислових середовищ.

Інтелектуальні методи аналізу - зокрема кластеризація апаратних метрик, розрахунок силуетного коефіцієнта та метод ліктя - дали змогу виокремити типові профілі навантаження та оцінити поведінку системи за різних умов. Це підтвердило релевантність використання машинного навчання для аналізу телеметрії та оптимізації налаштувань системи.

Комплексне тестування на основі розробленого плану включало функціональні, навантажувальні, стресові та довготривалі сценарії роботи. Отримані результати засвідчили здатність системи забезпечувати стабільну

інтенсивність потоків подій (до ≈300–320 подій/с у штатних режимах), низьку середню затримку (< 50 мс), мінімальні втрати (< 1 % для більшості профілів), а також прогнозовану деградацію при підвищеному навантаженні (256 вузлів). Проведений аналіз безпеки підтвердив надійність застосування TLS-шифрування, стійкість до нестабільності мережевого каналу та відповідність механізмів передачі подій вимогам IoT-інфраструктур.

Формування інсталяційного пакета - включно з програмними модулями, конфігураційними файлами, ML-моделями, Docker-маніфестами та документацією - створило передумови для CI/CD-інтеграції й швидкого розгортання системи. Це суттєво підвищує практичну цінність розробки та робить її придатною для використання як у дослідницьких, так і у промислових середовищах.

Узагальнюючи результати роботи, можна сформулювати такі підсумкові положення:

1. проведено системний аналіз предметної області IoT-телеметрії та визначено ключові вимоги до моделювання навантаження, безпеки та оброблення подій.
2. Розроблено багатошарову архітектуру емулятора й серверної частини, що включає модулі генерації, маршрутизації, логування, збору метрик, аналітики та моніторингу.
3. Формалізовано алгоритми роботи основних компонентів, створено повний комплект UML-моделей, що забезпечує структурованість і відтворюваність реалізації.
4. Реалізовано інтелектуальну підсистему аналізу навантаження, що застосовує машинне навчання та забезпечує кластеризацію й оцінювання ефективності системи.
5. Виконано всебічне тестування, яке підтвердило стабільність, масштабованість і відповідність системи заданим функціональним і нефункціональним вимогам.

6. Сформовано інсталяційний пакет, що забезпечує автоматизоване розгортання та інтеграцію системи у промислові або дослідницькі середовища.

Виконана кваліфікаційна робота доводить ефективність обраних архітектурних і технологічних рішень, демонструє практичну реалізованість інтелектуальної системи емуляції IoT-подій і підтверджує її здатність функціонувати як повноцінний інструмент для моделювання, аналізу та тестування навантаження у сучасних розподілених IoT-платформах. Система має значний потенціал для подальшого розвитку та інтеграції з реальними апаратними інфраструктурами, аналітичними сервісами й промисловими рішеннями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Таненбаум Е., Уезеролл Д. Комп'ютерні мережі. 5-те вид. Київ: Вид. група BHV, 2020. 912 с.

2. MQTT Version 3.1.1. OASIS Standard. OASIS, 2014. URL: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1

3. Національний університет біоресурсів і природокористування України. Методичні рекомендації щодо написання та оформлення кваліфікаційних робіт. Київ: НУБіП, 2021.

4. Love, R. Linux System Programming. O'Reilly Media, 2017. 456 p.

5. McKinney, W. Python for Data Analysis. 3rd ed. O'Reilly Media, 2022. 550 p.

6. Pedregosa, F. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2011, pp. 2825–2830.

7. Fielding, R. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, 2000.

8. RFC 6455. The WebSocket Protocol. IETF, 2011. URL: https://datatracker.ietf.org/doc/rfc6455/

9. Dean, J.; Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1), 2008, pp. 107–113.

10. Bishop, C. Pattern Recognition and Machine Learning. Springer, 2006. 738 p.

11. Kurose, J.; Ross, K. Computer Networking: A Top-Down Approach. 8th ed. Pearson, 2021.

12. ISO/IEC 20922:2016. Information Technology – Message Queuing Telemetry Transport (MQTT) v3.1.1. ISO, 2016.

13. OpenAPI Initiative. The OpenAPI Specification. Ver. 3.0. URL: https://www.openapis.org

14. Richardson, L.; Amundsen, M. RESTful Web APIs. O'Reilly Media, 2021.

15. Grafana Labs. Grafana Documentation. URL: https://grafana.com/docs

16. Prometheus Authors. Prometheus: Monitoring System & Time Series Database. URL: https://prometheus.io/docs

17. PostgreSQL Global Development Group. PostgreSQL 15 Documentation. URL: https://www.postgresql.org/docs

18. OSI. The Transport Layer Security (TLS) Protocol. RFC 8446 (TLS 1.3). IETF, 2018.

19. Barabanov A., Kurnikov A. High-Performance Event Processing in IoT Systems. *IEEE IoT Journal*, 2022.

20. Marwala, T. Artificial Intelligence and Signal Processing. Springer, 2023.