

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

**Навчально-науковий Інститут телекомунікаційних систем
Кафедра електронних комунікацій та інтернету речей**

«До захисту допущено»

Завідувач кафедри

_____ Анатолій МАКАРЕНКО

«__» _____ 20__ р.

Дипломна робота

на здобуття ступеня бакалавра

зі спеціальності 172 Телекомунікації та радіотехніка

на тему: « Реалізація рішення IoT за допомогою штучного інтелекту (AI)»

Виконав (-ла):

студент (-ка) IV курсу, групи ТС-21

Прібиш Стефан Андрійович _____

Керівник:

Доцент кафедри ЕКІР, доцент, к.т.н.

Осипчук Сергій Олександрович _____

Рецензент:

Доцент кафедри ТК НН ІТС, к.т.н. с.н.с.

Міночкін Дмитро Анатолійович _____

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без
відповідних посилань.

Студент (-ка) _____

Київ – 2026 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий Інститут телекомунікаційних систем
Кафедра електронних комунікацій та інтернету речей

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 172 Телекомунікації та радіотехніка

Освітня програма – «Системи електронних комунікацій та Інтернету речей»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Анатолій МАКАРЕНКО

«__» _____ 20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Прібишу Стефану Андрійовичу

1. Тема роботи « Реалізація рішення IoT за допомогою штучного інтелекту (AI) », керівник роботи Осипчук Сергій Олександрович, к.т.н., доцент, затверджені наказом по університету від «26» травня 2026 р. № 1912-С
2. Термін подання студентом роботи 13.06.2026
3. Вихідні дані до роботи: матеріали статей та наукових видань, інформаційні ресурси мережі Інтернет, навчально-методичні матеріали. Структурований план порядку розробки матеріалів дипломної роботи.
4. Зміст роботи: Обґрунтувати актуальність теми. Розглянути та проаналізувати сучасні технології побудови AIoT-систем та концепції Edge, Fog і Cloud Computing. Виконати порівняльний аналіз методів штучного інтелекту для задач класифікації станів обладнання та протоколів передачі даних для IoT. Спроектувати трирівневу архітектуру системи та обґрунтувати вибір компонентної бази на основі ESP32-S3. Реалізувати локальний інференс моделі MLP на периферійному вузлі та розробити імітаційну модель Fog-рівня для верифікації системи. Провести експериментальні дослідження та оцінити ефективність розробленої AIoT-системи.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо):
Рисунок 1 – Роль AIoT у предиктивному обслуговуванні промислового обладнання.
Рисунок 2 – Логіка досягнення мети дипломної роботи.

Рисунок 3 – Порівняння традиційної IoT-архітектури та запропонованого AIoT-підходу.

Рисунок 4 – Порівняння характеристик протоколів HTTP, MQTT, CoAP та LoRaWAN.

Рисунок 5 – Структурна схема запропонованої AIoT-системи Edge–Fog–Cloud.

Таблиця 1 – Порівняльна характеристика апаратних платформ для задач AIoT.

Рисунок 6 – Зображення ESP32.

Рисунок 7 – BME280.

Рисунок 8 – Алгоритм роботи AIoT-системи за моделлю Sense–Think–Act.

Рисунок 9 – Віртуальна схема Edge-вузла в середовищі моделювання Wokwi.

Рисунок 10 – Інтерфейс Wokwi.

Рисунок 11 – Послідовність обміну даними між Fog, Edge, MQTT-брокером і Cloud.

Таблиця 2 – Зведена статистика продуктивності AIoT-системи.

Рисунок 12 – Статистика експериментальних вимірювань.

Рисунок 13 – Налаштований веб-інтерфейс HiveMQ Web Client.

6. Дата видачі завдання 06.01.2026

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз технологій побудови систем IoT, методів AI та протоколів передачі даних	01.03.2026 – 20.03.2026	
2	Проектування архітектури AIoT-системи та вибір компонентної бази	21.03.2026 – 10.04.2026	
3	Практична реалізація: програмування Edge-вузлів та розробка імітаційної моделі Fog-рівня	11.04.2026 – 15.05.2026	
4	Проведення експериментальних досліджень та аналіз отриманих результатів	26.05.2026 – 30.05.2026	
5	Вступ, Висновки	31.05.2026	
6	Чистовий варіант дипломної роботи, плакати	01.06.2026	

Студент

Стефан ПРИБИШ

Керівник роботи

Сергій ОСИПЧУК

РЕФЕРАТ

Текстова частина дипломної роботи: 85 с., 7 рис., 3 табл., 34 джерел.

Актуальність роботи: зумовлена необхідністю переходу промислових підприємств до концепції Індустрії 4.0 та впровадження систем предиктивного обслуговування. Традиційні хмарні рішення часто не забезпечують необхідної швидкості реакції для критичних вузлів обладнання через мережеві затримки. Використання технології AIoT (Artificial Intelligence of Things) з розподіленою обробкою даних дозволяє виносити інтелектуальний аналіз безпосередньо на периферійні пристрої (Edge Computing), що значно підвищує автономність та швидкість детекції дефектів.

Мета роботи: розробка та практична реалізація інтелектуальної AIoT-системи для моніторингу та діагностики станів промислових підшипників у режимі реального часу з використанням ієрархічної архітектури Edge-Fog-Cloud.

Задачі дослідження: У роботі проведено аналіз сучасних технологій побудови AIoT-систем та протоколів передачі даних, зокрема концепцій Edge, Fog і Cloud Computing. На основі цього спроектовано трирівневу архітектуру Edge-Fog-Cloud, яка забезпечує ефективний розподіл обчислювального навантаження та мінімальні затримки при обробці телеметрії.

Обґрунтовано вибір мікроконтролера ESP32-S3 як апаратної основи системи та використання протоколу MQTT для передачі даних між вузлами AIoT-мережі. Реалізовано локальний інференс нейромережевої моделі MLP на периферійному вузлі із застосуванням MicroPython, що дозволило виконувати аналіз даних безпосередньо на Edge-рівні.

Для перевірки працездатності системи створено імітаційну модель сенсорного рівня на базі Fog-вузла з використанням датасету Case Western Reserve University Bearing Dataset. Проведені експериментальні дослідження підтвердили ефективність системи, її швидкодію та достатню точність виявлення аномалій у режимі реального часу.

Об'єкт дослідження: процеси моніторингу та інтелектуальної діагностики стану промислового обладнання в розподілених мережах Інтернету речей

Предмет дослідження: методи та засоби побудови AIoT-систем із застосуванням периферійних обчислень для автоматизованої класифікації станів підшипників кочення

Ключові слова: AIoT, EDGE COMPUTING, FOG COMPUTING, ПРЕДИКТИВНЕ ОБСЛУГОВУВАННЯ, ESP32, MICROPYTHON, MQTT, ДІАГНОСТИКА ПІДШИПНИКІВ

ABSTRACT

The work contains 85 pages, 7 figures, 3 tables, 34 sources.

Relevance of the work: The relevance is due to the necessity for industrial enterprises to transition to the Industry 4.0 concept and implement predictive maintenance systems. Traditional cloud solutions often fail to provide the required response speed for critical equipment components due to network latency. Utilizing AIoT (Artificial Intelligence of Things) technology with distributed data processing allows for moving intelligent analysis directly to peripheral devices (Edge Computing), significantly increasing autonomy and defect detection speed.

The purpose of the work: The purpose is the development and practical implementation of an intelligent AIoT system for real-time monitoring and diagnostics of industrial bearing states using a hierarchical Edge-Fog-Cloud architecture.

Research tasks: The work analyzes modern AIoT system construction technologies and data transmission protocols, specifically the concepts of Edge, Fog, and Cloud Computing. Based on this, a three-level Edge-Fog-Cloud architecture was designed to ensure efficient distribution of the computational load and minimal latency during telemetry processing. The choice of the ESP32-S3 microcontroller as the hardware foundation and the use of the MQTT protocol for data transmission between AIoT network nodes are justified. Local inference of the MLP neural network model was implemented on the edge node using MicroPython, enabling data analysis directly at the Edge level. To verify the system's performance, a sensor-level simulation model was created based on a Fog node using the Case Western Reserve University Bearing Dataset. Experimental studies confirmed the system's effectiveness, its performance speed, and sufficient accuracy in real-time anomaly detection.

Object of research: The processes of monitoring and intelligent diagnostics of industrial equipment status in distributed Internet of Things networks.

Subject of research: Methods and tools for building AIoT systems using edge computing for automated classification of rolling bearing states.

Keywords: AIoT, EDGE COMPUTING, FOG COMPUTING, PREDICTIVE MAINTENANCE, ESP32, MICROPYTHON, MQTT, BEARING DIAGNOSTICS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ ..	11
ВСТУП.....	14
1 АНАЛІЗ ТЕХНОЛОГІЙ ПОБУДОВИ СИСТЕМ ІОТ З ІНТЕЛЕКТУАЛЬНОЮ СКЛАДОВОЮ.....	16
1.1 Концепція інтелектуального моніторингу та предиктивного обслуговування в парадигмі Industry 4.0.....	16
1.2 Огляд архітектури сучасних систем Інтернету речей: Рівні сприйняття, мережевий рівень, рівень обробки даних (Cloud/Fog/Edge Computing).....	17
1.3 Аналіз методів штучного інтелекту для задач IoT: Машинне навчання для прогнозування, класифікації подій або виявлення аномалій у телеметрії.....	22
1.3.1 Обробка поточкових даних (Stream Processing) у AIoT	22
1.3.2 Машинне навчання для прогнозування (Forecasting/Regression)	23
1.3.3 Edge AI та розподілена інтелектуальна обробка	24
1.3.4 Класифікація подій та розпізнавання станів.....	25
1.3.5 Проблема дрейфу даних та адаптивність моделей	26
1.3.6 Виявлення аномалій (Anomaly Detection)	26
1.3.7 Ефективність та результати.....	27
1.3.8 Пояснюваний штучний інтелект (Explainable AI).....	27
1.4 Порівняльна характеристика протоколів передачі даних: MQTT, CoAP, HTTP, LoRaWAN.....	28
1.4.1 HTTP (Hypertext Transfer Protocol)	28
1.4.2 MQTT (Message Queuing Telemetry Transport)	29
1.4.3 CoAP (Constrained Application Protocol).....	30
1.4.4 LoRaWAN (Long Range Wide Area Network).....	30
1.5 Постановка задачі на дипломну роботу: Обґрунтування необхідності створення власного рішення (наприклад, для розумного будинку, агросектору чи моніторингу обладнання).	32
1.6 Висновки до розділу 1	34

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ВИБІР КОМПОНЕНТНОЇ БАЗИ СИСТЕМИ	36
2.1 Розробка структурної схеми системи: Взаємодія вузлів (сенсори - мікроконтролер -шлюз -сервер/хмара).....	36
2.1.1 Обґрунтування та деталізація топології Edge-Fog-Cloud.....	36
2.1.2 Функціональне призначення вузлів системи.....	38
2.1.3 Логіка обміну повідомленнями та потоки даних	41
2.2 Обґрунтування вибору апаратного забезпечення.....	43
2.2.1 Вибір мікроконтролера з точки зору обчислювальної потужності для AI	43
2.2.2 Детальний технічний аналіз платформ	45
2.2.3 Вибір датчиків та виконавчих механізмів.	46
2.3 Вибір середовища розробки та програмних інструментів	50
2.3.1 Вибір середовища розробки та програмних інструментів	51
2.3.2 Мови програмування.....	52
2.4 Проєктування алгоритму роботи системи: Блок-схема логіки прийняття рішень на основі даних AI.....	55
2.5 Висновки до розділу 2	58
3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ AIOT-СИСТЕМИ.....	60
3.1 Організація середовища розробки та інструментарій моделювання.....	60
3.1.1 Налаштування інтегрованого середовища Visual Studio Code та інструментів розробки.....	61
3.1.2 Використання хмарної платформи Wokwi для віртуального прототипування Edge-вузлів.....	62
3.1.3 Опис апаратних зв'язків у форматі коду.....	63
3.2 Програмна реалізація інтелектуального вузла на базі MicroPython.....	64
3.2.1 Алгоритмічне забезпечення обробки ознак на ESP32.....	64
3.2.2 Інтеграція OLED-інтерфейсу для візуалізації станів системи.....	65
3.3 Розробка імітаційної моделі Fog-рівня (Python-симулятор)	66
3.3.1 Архітектура та логіка роботи цифрового двійника	66

	10
3.3.2 Механізм зворотного зв'язку та аналізу метрик	67
3.4 Аналіз результатів експериментальних досліджень	68
3.5 Верифікація даних через хмарний інтерфейс моніторингу.....	69
3.5.1 Використання хмарного MQTT-клієнта для візуалізації	70
3.5.2 Аналіз структури та синхронності JSON-пакетів	71
3.6 Висновки до розділу 3	71
ВИСНОВКИ.....	74
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	78
ДОДАТОК А.....	83
ДОДАТОК Б	84
ДОДАТОК В	85

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

AIoT	Artificial Intelligence of Things - штучний інтелект речей; концепція об'єднання IoT та III для створення інтелектуальних систем
AI	Artificial Intelligence - штучний інтелект
API	Application Programming Interface - інтерфейс програмування застосунків
CNN	Convolutional Neural Network - згортова нейронна мережа
CoAP	Constrained Application Protocol - протокол для пристроїв з обмеженими ресурсами
CWRU	Case Western Reserve University - університет, чий датасет підшипників використано в роботі
DSP	Digital Signal Processing - цифрова обробка сигналів
E2E	End-to-End - наскрізний (від кінця до кінця)
Edge Computing	Периферійні обчислення - обробка даних безпосередньо поблизу джерела даних
ESP32	Мікроконтролер компанії Espressif з інтегрованими модулями Wi-Fi та Bluetooth
Fog Computing	Туманні обчислення - проміжний рівень між Edge та Cloud для агрегації та аналізу даних
FPGA	Field-Programmable Gate Array - програмована логічна інтегральна схема
GPIO	General Purpose Input/Output - порти загального призначення мікроконтролера
HTTP	Hypertext Transfer Protocol - протокол передачі гіпертексту
I2C	Inter-Integrated Circuit - двопровідний послідовний інтерфейс зв'язку
IDE	Integrated Development Environment - інтегроване

середовище розробки

IoT	Internet of Things - Інтернет речей
IP	Internet Protocol - протокол Інтернету
JSON	JavaScript Object Notation - текстовий формат обміну даними
LoRaWAN	Long Range Wide Area Network - протокол мережі дальнього радіуса дії з низьким споживанням
LPWAN	Low Power Wide Area Network - мережа дальнього радіуса дії з низьким споживанням
LSTM	Long Short-Term Memory - різновид рекурентної нейронної мережі для роботи з послідовностями
MAC	Media Access Control - адреса пристрою на каналному рівні мережі
MCU	Microcontroller Unit - мікроконтролер
MDE	Model-Driven Engineering - підхід розробки програмного забезпечення на основі моделей
MLP	Multilayer Perceptron - багатошаровий перцептрон; тип нейронної мережі прямого поширення
MPU-6050	Шестиосьовий датчик акселерометра та гіроскопа з інтерфейсом I2C (InvenSense)
MQTT	Message Queuing Telemetry Transport - легкий протокол обміну повідомленнями для IoT
OLED	Organic Light-Emitting Diode - органічний світлодіодний дисплей
QoS	Quality of Service - якість обслуговування; рівень надійності доставки повідомлень
RAM	Random Access Memory - оперативна пам'ять
REST	Representational State Transfer - архітектурний стиль побудови API на основі HTTP
RMS	Root Mean Square - середньоквадратичне значення

RNN	Recurrent Neural Network - рекурентна нейронна мережа
STM32	Серія 32-розрядних мікроконтролерів компанії STMicroelectronics на базі ARM Cortex-M
TCP	Transmission Control Protocol - протокол управління передачею з гарантією доставки
TinyML	Галузь машинного навчання, орієнтована на розгортання ML-моделей на периферійних пристроях
UDP	User Datagram Protocol - протокол передачі даних без гарантії доставки
ZigBee	Протокол бездротового зв'язку для персональних мереж малого радіуса дії
MH	Машинне навчання
ШІ	Штучний інтелект

ВСТУП

В умовах сучасного розвитку промисловості перехід підприємств до концепції Індустрії 4.0 та систем предиктивного обслуговування перетворився на ключовий елемент забезпечення безперебійного виробництва. Однак ефективність традиційних рішень Інтернету речей (IoT) напряду залежить від стабільності підключення до хмарної інфраструктури, яка часто не забезпечує необхідної швидкості реакції для критичних вузлів обладнання через мережеві затримки. Передача масивів сирих даних на віддалені сервери може призвести не тільки до перевантаження каналів, але й до запізненої реакції на аварійні стани.

У відповідь на ці виклики виникає гостра необхідність у розробці та впровадженні передових рішень, здатних перенести обчислювальні потужності ближче до джерела даних. Сучасні технології, такі як штучний інтелект речей (AIoT) та периферійні обчислення (Edge Computing), відкривають нові можливості для створення автономних і швидких систем, здатних виконувати інтелектуальний аналіз та приймати рішення безпосередньо на периферійних пристроях.

Метою цієї роботи є розробка та практична реалізація інтелектуальної AIoT-системи для моніторингу та діагностики станів промислових підприємств у режимі реального часу з використанням ієрархічної архітектури Edge-Fog-Cloud. Для досягнення цієї мети поставлено такі задачі: проаналізувати сучасні технології побудови AIoT-систем та концепції розподілених обчислень, спроектувати трирівневу архітектуру для ефективного розподілу обчислювального навантаження, обґрунтувати вибір мікроконтролера ESP32-S3 та протоколу MQTT, реалізувати локальний інференс нейромережевої моделі на периферійному вузлі та, на основі отриманих даних, розробити імітаційну модель для верифікації системи.

Об'єктом дослідження є процеси моніторингу та інтелектуальної діагностики стану промислового обладнання в розподілених мережах Інтернету речей. Предметом дослідження виступають методи та засоби побудови AIoT-

систем із застосуванням периферійних обчислень для автоматизованої класифікації станів підшипників кочення.

1 АНАЛІЗ ТЕХНОЛОГІЙ ПОБУДОВИ СИСТЕМ ІОТ З ІНТЕЛЕКТУАЛЬНОЮ СКЛАДОВОЮ

1.1 Концепція інтелектуального моніторингу та предиктивного обслуговування в парадигмі Industry 4.0

Сучасний етап розвитку інформаційних технологій характеризується глибокою конвергенцією Інтернету речей та штучного інтелекту, що формує нову парадигму -AIoT. На відміну від традиційних систем, де пристрої виконують лише функції збору та передачі даних, інтелектуальна складова дозволяє системі самостійно аналізувати інформацію, приймати рішення та адаптуватися до змін навколишнього середовища без постійного втручання людини [1]. Побудова таких систем вимагає комплексного підходу до архітектури, де ключову роль відіграють рівні сенсорики, обчислень та мережевої взаємодії [2].

В архітектурному плані виділяють перехід від хмаро-орієнтованих систем до периферійних та туманних обчислень. Це критично для систем, що потребують миттєвої реакції, оскільки обробка даних безпосередньо поблизу джерела на сенсорах або шлюзах дозволяє значно знизити затримки та розвантажити канали зв'язку. Використання методів машинного навчання на периферійних пристроях дає змогу реалізувати функції прогнозування майбутнього стану системи на основі історичних даних, що підвищує загальну автономність [3]. Зокрема, для аналізу динамічних середовищ та складних паттернів руху застосовуються згорткові та рекурентні нейронні мережі (RNN), які довели свою ефективність у задачах класифікації об'єктів та виявлення аномалій у реальному часі [4].

Важливою складовою аналізу є технології передачі даних, які забезпечують зв'язність інтелектуальних вузлів. Для реалізації надійного обміну інформацією використовуються енергоефективні мережі дальнього радіуса дії, що дозволяють передавати дані на відстані до 10-15 км при мінімальному споживанні енергії. На програмному рівні взаємодія пристроїв часто базується на протоколах MQTT та CoAP, які оптимізовані для обмежених ресурсів сенсорних мереж та підтримують механізми забезпечення якості обслуговування (QoS) [5].

Особлива увага при проектуванні систем з інтелектуальним компонентом приділяється методам оптимізації функціонування. Впровадження алгоритмів МН дозволяє не лише аналізувати дані, а й керувати ресурсами самої системи: оптимізувати енергоспоживання вузлів та динамічно змінювати параметри передачі сигналів в умовах завад [3]. Це відкриває можливості для створення високонадійних рішень у сферах інтелектуального транспорту, моніторингу критичної інфраструктури та автономних робототехнічних комплексів [6]. Сучасні дослідження підкреслюють, що інтеграція ШІ в IoT-мережі є ключовим фактором для переходу до концепцій Індустрії 4.0 та побудови розумних міст.

З точки зору безпеки, інтелектуальна складова стає фундаментом для виявлення кіберзагроз на ранніх стадіях. Аналіз трафіку за допомогою нейронних мереж дозволяє ідентифікувати несанкціоноване втручання або спроби підміни даних у радіоканалах, що є критичним для систем, де цілісність інформації безпосередньо впливає на безпеку процесів [1, 3]. Таким чином, сучасні технології побудови AIoT-систем поєднують у собі передові апаратні рішення, високоефективні протоколи зв'язку та глибоку аналітику даних для створення повністю автономних та захищених екосистем.

1.2 Огляд архітектури сучасних систем Інтернету речей: Рівні сприйняття, мережевий рівень, рівень обробки даних (Cloud/Fog/Edge Computing)

Сучасна архітектура систем Інтернету речей зазнала трансформації від простих структур збору даних до інтелектуальних екосистем, де кожен рівень виконує специфічні функції для забезпечення автономності та ефективності. Основна мета такої архітектури полягає в інтеграції фізичного світу з цифровим через безперервний цикл збору, передачі та аналізу інформації [4]. Найбільш поширеною є трирівнева модель, яка включає рівень сприйняття, мережевий рівень та рівень обробки даних, що разом формують фундамент для систем типу AIoT [2].

1.2.1 Рівень сприйняття (Perception Layer)

Цей рівень є первинною ланкою взаємодії системи з фізичним середовищем. Його основна функція полягає в ідентифікації об'єктів та зборі фізичних параметрів за допомогою датчиків (температури, вологості, тиску, відеокамер) та виконавчих механізмів [5]. У сучасних інтелектуальних системах рівень сприйняття стає «активним». Замість простої ретрансляції сигналів, вузли сприйняття все частіше виконують первинну обробку даних безпосередньо «на борту».

Використання програмованих логічних інтегральних схем (FPGA) та спеціалізованих мікроконтролерів дозволяє реалізувати методи машинного навчання для фільтрації шумів та стиснення даних безпосередньо на сенсорі [3]. Це критично важливо для зменшення обсягу надлишкової інформації, що передається в мережу, та зниження енергоспоживання автономних пристроїв [4]. Таким чином, рівень сприйняття забезпечує не лише збір даних, а й початкову інтелектуальну селекцію сигналів, що підвищує загальну продуктивність системи [6].

1.2.2 Мережевий рівень (Network Layer)

Мережевий рівень виконує роль комунікаційного фундаменту, забезпечуючи надійне з'єднання між фізичними пристроями та хмарними або периферійними центрами обробки. Він охоплює широкий спектр технологій бездротового зв'язку, які обираються залежно від конкретних вимог до дальності та енергоефективності. У локальних системах автоматизації часто використовуються протоколи малого радіуса, такі як ZigBee або Bluetooth LE, тоді як для територіально розподілених мереж пріоритетними є технології LPWAN, зокрема LoRaWAN, що дозволяє передавати дані на відстань до 10-15 км [5].

Важливою характеристикою сучасного мережевого рівня є використання спеціалізованих стеків протоколів, таких як MQTT та CoAP. Ці протоколи

оптимізовані для роботи в умовах обмежених ресурсів та нестабільного зв'язку, що характерно для IoT-мереж [5]. Окрім транспортування даних, мережевий рівень відповідає за агрегацію трафіку та забезпечення безпеки передачі. В умовах інтелектуальних систем цей рівень також може задіяти алгоритми ШІ для динамічної зміни параметрів зв'язку з метою захисту від завад та оптимізації маршрутизації в реальному часі [6, 4].

1.2.3 Рівень обробки даних (Processing Layer)

Рівень обробки даних є «інтелектуальним центром» системи, де інформація перетворюється на знання та керуючі команди. У сучасних архітектурах спостерігається перехід від централізованих хмарних обчислень до гібридних моделей, що включають периферійні (Edge) та туманні (Fog) обчислення [2]. Периферійні обчислення дозволяють виконувати складні алгоритми машинного навчання безпосередньо поблизу джерел даних, що мінімізує затримки (latency) та дозволяє системі реагувати на події в режимі реального часу [3]. Уточнена модель обробки даних включає три ключові рівні, кожен з яких виконує специфічні завдання в межах загальної методології моніторингу та управління [7, 13].

1.2.4 Ієрархія Cloud/Fog/Edge

Периферійні обчислення (Edge Computing) становлять нижній рівень ієрархії, де обробка відбувається безпосередньо на кінцевих пристроях або інтелектуальних сенсорах. На цьому рівні реалізується концепція «обробки на місці» (in-situ), що дозволяє виконувати первинну фільтрацію телеметрії, стиснення даних та виявлення миттєвих критичних станів [7]. Використання оптимізованих моделей машинного навчання (TinyML) на рівні Edge забезпечує реакцію системи протягом мілісекунд, що є вирішальним для автономних об'єктів та систем безпеки. Окрім того, периферійна обробка значно знижує навантаження

на комунікаційні канали, відсікаючи надлишкову інформацію ще до її потрапляння в мережу.

Туманні обчислення (Fog Computing) виступають проміжною ланкою, що агрегує дані від групи периферійних вузлів. Fog-вузли (інтелектуальні шлюзи або локальні сервери) забезпечують координацію між пристроями та виконують складніші аналітичні завдання, які потребують контексту від декількох джерел одночасно [7]. Згідно з методиками проектування центрів збирання та оброблення даних, туманний рівень відіграє критичну роль у забезпеченні цілісності моніторингу об'єктів критичної інфраструктури. Він дозволяє системі зберігати працездатність та виконувати алгоритми управління навіть у разі повної втрати зв'язку з центральною хмарою, виступаючи локальним центром прийняття рішень.

Хмарні обчислення (Cloud Computing) займають верхній рівень ієрархії та відповідають за глобальну аналітику, довгострокове зберігання «великих даних» (Big Data) та навчання складних нейромережових моделей [7]. Саме в хмарі відбувається формування стратегічних прогнозів та навчання інтелектуальної складової на основі масивів даних, зібраних з усієї мережі об'єктів. Методологія формування структури центрів оброблення передбачає, що хмарний рівень забезпечує високу масштабованість та доступ до потужних обчислювальних ресурсів, які неможливо розгорнути локальн.

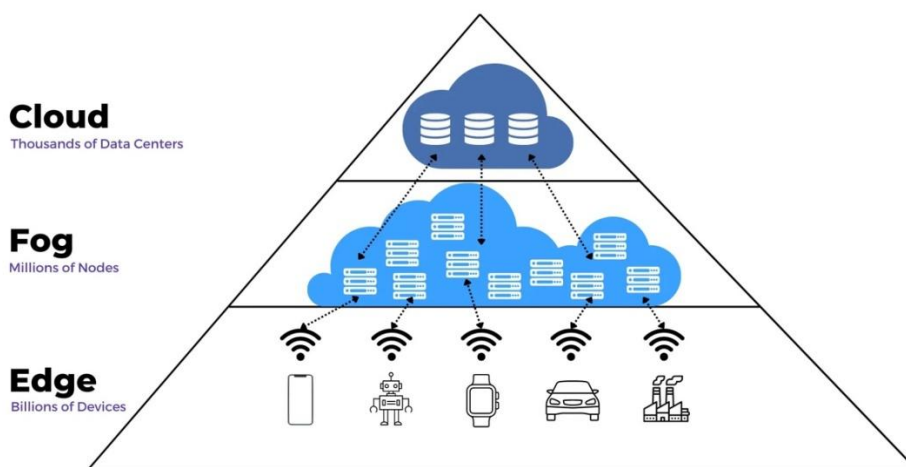


Рисунок 1.1 — Ієрархічна модель розподілених обчислень Cloud, Fog та Edge у системах IoT [8]

На рисунку 1.1 представлена дворівнева ієрархія обробки даних, що демонструє розподіл обчислювальних ресурсів у сучасних AIoT-системах. Модель побудована у формі піраміди, що відображає залежність між обсягом даних, швидкістю реакції та обчислювальною потужністю на кожному рівні:

Рівень периферійних обчислень (Edge Layer): Знаходиться в основі піраміди та охоплює кінцеві інтелектуальні пристрої та сенсори. Цей рівень забезпечує обробку даних у реальному часі з мінімальною затримкою (мілісекунди), що необхідно для миттєвого реагування на критичні події телеметрії.

Рівень туманних обчислень (Fog Layer): Виступає проміжною ланкою, що складається з інтелектуальних шлюзів та локальних серверів. Він забезпечує агрегацію даних з декількох Edge-вузлів, локальне зберігання та ширший контекстний аналіз, гарантуючи живучість системи при відсутності зв'язку з глобальною мережею.

Рівень хмарних обчислень (Cloud Layer): Вершина піраміди, де зосереджені найбільші обчислювальні потужності. Тут виконується глибоке навчання нейромережових моделей на масивах Big Data та стратегічне прогнозування. Хоча цей рівень має найбільшу затримку, він забезпечує найвищий рівень інтелектуальної обробки.

Синергія рівнів та методологічний підхід Ефективність такої ієрархії досягається через динамічний розподіл обчислювального навантаження. Застосування методики формування структур центрів оброблення даних дозволяє оптимізувати потоки телеметрії, забезпечуючи баланс між швидкістю реакції на периферії та глибиною аналізу в центрі. Це створює стійку до відмов архітектуру, де інтелектуальна складова розподілена по всій системі, що є обов'язковою умовою для сучасних рішень у сферах енергетики, транспорту та захисту об'єктів критичної інфраструктури.

Для аналізу отриманих даних використовуються передові методи глибинного навчання. Наприклад, згорткові нейронні мережі застосовуються для обробки візуальної інформації та розпізнавання образів, а рекурентні нейронні мережі, зокрема BiLSTM, ефективні для аналізу часових рядів та прогнозування

стану системи [1]. Завдяки інтеграції інтелектуальної складової на рівні обробки, система отримує здатність до самонавчання та предиктивного керування, що є ключовим фактором для впровадження рішень у медицині, промисловості та концепції «розумного міста» [2, 6].

1.3 Аналіз методів штучного інтелекту для задач IoT: Машинне навчання для прогнозування, класифікації подій або виявлення аномалій у телеметрії

У сучасних високонавантажених інфраструктурах IoT традиційні методи моніторингу, що покладаються на порогові сповіщення та ретроспективний аналіз, виявляються недостатньо ефективними через динамічну складність середовищ [9]. Інтелектуальна складова відіграє ключову роль у переході до парадигми «проактивної спостережуваності» (observability). Використання алгоритмів штучного інтелекту (ШІ) дозволяє не просто фіксувати стан системи, а й аналізувати багатовимірні потоки телеметрії для виявлення ознак деградації продуктивності до того, як відбудеться порушення рівня обслуговування (SLA) [9, 10].

Інтеграція ШІ з Інтернетом речей забезпечує можливість обробки високочастотних даних у реальному часі, що є критично важливим для промислового IoT. Інтелектуальні компоненти дозволяють автоматизувати процес прийняття рішень, мінімізувати вплив людського фактору та забезпечувати високу надійність систем в умовах великих даних [10, 11].

1.3.1 Обробка поточкових даних (Stream Processing) у AIoT

Особливістю IoT-систем є безперервна генерація поточкових даних великого обсягу, що вимагає використання алгоритмів машинного навчання, здатних працювати у режимі реального часу. На відміну від класичних інформаційних систем, де аналіз часто виконується постфактум, AIoT-платформи повинні забезпечувати миттєву реакцію на зміну стану об'єкта або середовища [11]. У

зв'язку з цим широкого застосування набувають технології потокової аналітики (Stream Analytics), інтегровані з ML-моделями.

Для обробки телеметричних потоків використовуються Apache Kafka, Apache Flink, Spark Streaming та TensorFlow Extended (TFX), які дозволяють реалізовувати масштабовані конвеєри аналітики даних. Такі архітектури забезпечують безперервне навчання моделей, автоматичне оновлення параметрів та оперативне виявлення аномальних подій навіть у високонавантажених промислових мережах [10].

Особливе значення потокова аналітика має у критичних IoT-системах, зокрема в енергетиці, транспорті, системах автономного керування та промисловій автоматизації, де затримка у прийнятті рішення навіть на декілька секунд може призвести до значних технічних або економічних втрат.

1.3.2 Машинне навчання для прогнозування (Forecasting/Regression)

Прогнозування на основі машинного навчання є фундаментом для реалізації стратегій прогнозного обслуговування (Predictive Maintenance), спрямованих на максимізацію часу безвідмовної роботи та оптимізацію життєвого циклу активів [11, 12].

Згідно з аналізом сучасних методів, для прогнозування в системах IoT застосовуються наступні підходи:

Архітектури на основі Трансформерів: Ці моделі демонструють перевагу в аналізі часових рядів телеметрії завдяки механізмам уваги, які дозволяють ефективно обробляти довгострокові залежності та нелінійні патерни у багатовимірних даних [13]. Трансформери забезпечують високу точність прогнозування навіть в екстремальних умовах експлуатації, де традиційні статистичні моделі або прості рекурентні мережі не справляються зі складністю даних [13].

Глибоке навчання (CNN та LSTM): Комбінація згорткових нейронних мереж (CNN) та мереж довгої короткострокової пам'яті (LSTM) активно

використовується для передбачення відмов обладнання та оцінки його залишкового ресурсу. CNN дозволяють витягувати просторові ознаки з даних датчиків, а LSTM -враховувати часову послідовність подій [10].

Класичні алгоритми машинного навчання: Для задач IoT часто застосовуються такі методи, як випадковий ліс (Random Forest), градієнтний бустинг (Gradient Boosting) та машини опорних векторів (SVM). Вони цінуються за їхню обчислювальну ефективність, масштабованість та здатність забезпечувати високу точність на різних промислових наборах даних при відносно низьких витратах ресурсів [11].

Прогностичне моделювання поведінки систем: Використання мультиваріативного прогнозування часових рядів у поєднанні з адаптивними петлями зворотного зв'язку дозволяє моделювати поведінку системи в реальному часі. Експериментальні дані показують, що такі інтелектуальні фреймворки можуть покращити показники надійності та передбачуваності системи на 27% [9].

1.3.3 Edge AI та розподілена інтелектуальна обробка

Однією з ключових тенденцій розвитку AIoT є перенесення алгоритмів штучного інтелекту безпосередньо на периферійні пристрої (Edge AI). Традиційна централізована модель, за якої всі дані передаються до хмарної інфраструктури для аналізу, створює значне навантаження на мережу та збільшує затримки передачі [12].

Використання Edge AI дозволяє виконувати попередню обробку, класифікацію та виявлення аномалій безпосередньо на IoT-вузлах або шлюзах. Це забезпечує зниження затримок, підвищення автономності системи та зменшення обсягу переданого трафіку. Особливо актуальним такий підхід є для систем з обмеженими часовими вимогами, наприклад безпілотних платформ, автономних робототехнічних систем та промислових сенсорних мереж [11].

Для реалізації Edge AI застосовуються оптимізовані моделі машинного навчання, зокрема TensorFlow Lite, TinyML та ONNX Runtime. Ці технології

дозволяють запускати нейронні мережі навіть на мікроконтролерах з обмеженими обчислювальними ресурсами та енергоспоживанням.

1.3.4 Класифікація подій та розпізнавання станів

Класифікація станів у системах АІoТ є критичним етапом, що дозволяє диференціювати нормальні робочі цикли від потенційно небезпечних режимів експлуатації. На відміну від простих порогових систем, інтелектуальні алгоритми здатні розпізнавати складні стани обладнання на основі багатовимірних сигналів від багатьох датчиків одночасно [10, 11].

Класичні методи навчання з учителем: Для задач промислового ІoТ широке застосування знаходять алгоритми Random Forest та Gradient Boosting. Вони демонструють високу обчислювальну ефективність та масштабованість, що дозволяє використовувати їх для класифікації станів активів у реальному часі навіть за умови великих обсягів вхідної телеметрії. Метод машин опорних векторів виявляється особливо дієвим при роботі з багатовимірними наборами даних, забезпечуючи чітке розмежування між «здоровим» та «несправним» станами об'єктів [11].

Глибоке навчання та витягування ознак: Використання згорткових нейронних мереж дозволяє автоматично витягувати просторові ознаки з даних телеметрії. У поєднанні з Big Data стеками (такими як Apache Spark), ці моделі здатні класифікувати складні патерни поведінки систем, забезпечуючи високу точність розпізнавання навіть за наявності значного мережевого шуму.

Роль у прогностному обслуговуванні (PdM): Класифікація є основою для визначення залишкового корисного ресурсу (RUL). Алгоритми ідентифікують специфічні «підписи» відмов, що дозволяє системі не просто фіксувати факт несправності, а й визначати її тип (наприклад, деградація підшипника чи збій електроніки), що значно спрощує подальше технічне обслуговування [10, 12].

1.3.5 Проблема дрейфу даних та адаптивність моделей

Однією з важливих проблем експлуатації AIoT-систем є дрейф даних (Data Drift) та дрейф концепції (Concept Drift). У реальних умовах характеристики телеметрії можуть змінюватися через фізичне старіння обладнання, зміну умов експлуатації, вплив середовища або оновлення програмного забезпечення [10].

У результаті модель машинного навчання, яка демонструвала високу точність на етапі навчання, поступово втрачає ефективність під час тривалої експлуатації. Для вирішення цієї проблеми використовуються механізми онлайн-навчання (Online Learning), адаптивного перенавчання та федеративного навчання (Federated Learning), що дозволяють оновлювати параметри моделей без повного повторного тренування системи.

Федеративне навчання є особливо перспективним для IoT, оскільки дозволяє навчати глобальну модель на розподілених пристроях без передачі сирих даних у центральне сховище. Це не лише знижує мережеве навантаження, але й підвищує рівень конфіденційності та кібербезпеки системи [12].

1.3.6 Виявлення аномалій (Anomaly Detection)

Виявлення аномалій у телеметрії -це процес ідентифікації відхилень від нормальної поведінки системи, які можуть передувати критичним збоям або порушенням безпеки. Сучасні фреймворки спостережуваності (observability) використовують нейронні класифікатори аномалій для проактивного запобігання деградації продуктивності [9].

Методи та архітектури:

Автоенкодери (Autoencoders): Цей підхід базується на принципі реконструкції вхідних даних. Модель навчається стискати та відновлювати нормальні дані телеметрії. Якщо на вхід подається аномальний сигнал, помилка реконструкції суттєво зростає.

Трансформери для екстремальних умов: Архітектури на основі Transformers використовують механізми самоуваги (Self-Attention) для виявлення нелінійних аномальних патернів, які часто губляться у традиційних рекурентних мережах. Це особливо актуально для складних систем, наприклад, систем життєзабезпечення космічних апаратів, де телеметрія містить довгі залежності та високу варіативність [13].

Мультиваріативна спостережуваність: Використання мультиваріативного прогнозування часових рядів у поєднанні з адаптивними петлями зворотного зв'язку дозволяє виявляти аномалії не лише за абсолютними значеннями параметрів, а й за порушенням кореляційних зв'язків між ними [9].

1.3.7 Ефективність та результати

Експериментальна валідація інтелектуальних систем виявлення аномалій демонструє значний приріст надійності. Впровадження нейронних класифікаторів та розподіленого трасування дозволяє виявляти деградацію системи задовго до порушення угод про рівень обслуговування, що в середньому покращує передбачуваність поведінки інфраструктури на 27% [9]. Окрім точності, велика увага приділяється масштабованості та здатності алгоритмів працювати в умовах дрейфу моделі (model drift), що забезпечує стабільність аналітики протягом тривалого часу [10, 11].

1.3.8 Пояснюваний штучний інтелект (Explainable AI)

У промислових та критичних IoT-системах важливим аспектом є не лише точність прогнозування, а й можливість інтерпретації рішень моделі. Складні нейронні мережі часто функціонують як «чорна скринька», що ускладнює аналіз причин прийняття рішення [11].

Для підвищення довіри до AIoT-систем використовуються підходи Explainable AI (XAI), які дозволяють визначити, які саме параметри телеметрії

найбільше вплинули на класифікацію події або виявлення аномалії. Найбільш поширеними методами є SHAP (SHapley Additive exPlanations), LIME (Local Interpretable Model-agnostic Explanations) та механізми attention-візуалізації у трансформерних архітектурах.

Використання Explainable AI є особливо важливим у системах критичної інфраструктури, де помилкові рішення алгоритмів можуть призвести до серйозних наслідків, а оператор повинен мати можливість перевірити логіку роботи інтелектуальної системи

Таким чином, сучасні методи штучного інтелекту в AIoT-системах забезпечують не лише автоматизований аналіз телеметричних потоків, а й формують основу для створення адаптивних, самонавчальних та проактивних систем моніторингу. Поєднання технологій машинного навчання, потокової аналітики, Edge AI та Explainable AI дозволяє значно підвищити ефективність прогнозування, точність класифікації станів та швидкість виявлення аномалій у складних розподілених інфраструктурах IoT.

1.4 Порівняльна характеристика протоколів передачі даних: MQTT, CoAP, HTTP, LoRaWAN.

Вибір комунікаційного протоколу є одним із найбільш критичних етапів проектування систем Інтернету речей, оскільки він безпосередньо впливає на архітектуру мережі, термін служби автономних пристроїв та надійність доставки телеметрії [14]. Сучасні протоколи розрізняються за архітектурними моделями, транспортними протоколами (TCP або UDP) та обсягом службової інформації (overhead), що додається до кожного повідомлення [15].

1.4.1 HTTP (Hypertext Transfer Protocol)

Хоча HTTP є найбільш поширеним протоколом у світі вебу, його використання в IoT обмежене через значні накладні витрати.

Архітектура: Базується на моделі «запит-відповідь», де клієнт ініціює кожен сеанс зв'язку. Це вимагає від пристрою постійного перебування в активному стані або частого пробудження для перевірки наявності команд [14].

Енергоефективність: HTTP використовує текстовий формат заголовків, розмір яких може сягати 700-800 байтів, що у сотні разів перевищує розмір корисного навантаження датчиків [14]. Крім того, робота поверх TCP вимагає процедури «трьохстороннього рукошлякування» (three-way handshake) для кожного з'єднання, що створює велике навантаження на батарею [15].

Сценарій використання: Підходить лише для пристроїв із постійним живленням, де важлива сумісність із існуючими веб-серверами та RESTful сервісами [14, 16].

1.4.2 MQTT (Message Queuing Telemetry Transport)

MQTT наразі є стандартом де-факто для систем моніторингу в реальному часі завдяки своїй асинхронній природі.

Механізм роботи: Використовує модель «публікація-підписка» за допомогою центрального вузла (Broker). Це дозволяє пристроям публікувати дані лише тоді, коли відбуваються зміни, не чекаючи запиту від сервера [16].

Обсяг даних: Протокол має двійковий заголовок розміром лише 2 байти, що робить його надзвичайно ефективним для вузькосмугових каналів [14].

Надійність (QoS): MQTT підтримує три рівні якості обслуговування (0 -без гарантії, 1 -доставка хоча б один раз, 2 -доставка рівно один раз), що дозволяє гнучко налаштовувати баланс між надійністю та споживанням трафіку [15]. Однак, використання TCP для підтримання постійної сесії («Keep Alive») вимагає певних енерговитрат, що робить його менш ефективним за CoAP для пристроїв, які більшу частину часу перебувають у режимі сну [14].

1.4.3 CoAP (Constrained Application Protocol)

CoAP було створено як спеціалізовану альтернативу HTTP для пристроїв із вкрай обмеженими ресурсами (RAM, Flash та живлення).

Архітектура: Як і HTTP, він підтримує REST-методи (GET, POST), але працює поверх UDP. Це дозволяє уникнути складних процедур встановлення та підтримання TCP-сесії [14].

Енергоефективність: CoAP використовує стиснений бінарний заголовок (4 байти) і дозволяє пристроям миттєво переходити в режим глибокого сну після надсилання повідомлення [15].

Функції: Протокол підтримує багатоадресну розсилку (multicast), що важливо для групового управління датчиками в автономних мережах [14].

1.4.4 LoRaWAN (Long Range Wide Area Network)

LoRaWAN стоїть окремо, оскільки це протокол мережевого рівня для енергоефективних мереж дальнього радіуса дії (LPWAN).

Дальність та споживання: Це лідер за показниками енергозбереження. Пристрої можуть працювати від однієї батареї типу AA до 10 років, передаючи дані на відстані до 15 км у сільській місцевості [15].

Пропускна здатність: LoRaWAN має дуже низьку швидкість передачі (від 0.3 до 50 кбіт/с), тому він не призначений для потокової передачі даних. Його основне завдання - періодична відправка коротких пакетів телеметрії [15].

Безпека: Протокол має вбудоване наскрізне шифрування AES-128, що робить його безпечним для промислового використання [15].

Таблиця 1.1 - Порівняльна таблиця характеристик протоколів

Характеристика	HTTP	MQTT	CoAP	LoRaWAN
Модель взаємодії	Запит-відповідь	Публікація-підписка	Запит-відповідь	Star Topology

Продовження таблиці 1.1

Розмір заголовка	Великий (~800 В)	Малий (2 В)	Малий (4 В)	Малий
Енергоефективність	Низька	Середня	Висока	Дуже висока
Гарантія доставки (QoS)	На рівні TCP	3 рівні (QoS 0, 1, 2)	Підтверджені/непідтв	На рівні MAC
Дальність дії	Залежить від IP-мережі	Залежить від IP-мережі	Залежить від IP-мережі	Велика (до 15 км)

Інтернету речей не існує універсального протоколу, який би одночасно задовольняв вимоги щодо максимальної дальності, мінімального енергоспоживання та високої пропускної здатності. Вибір конкретного стека технологій є результатом компромісу між технічними обмеженнями кінцевих пристроїв та бізнес-вимогами до функціонування системи.

На основі детального оцінювання характеристик можна виділити наступні сценарії вибору:

Максимальна енергонезалежність та радіус дії (LoRaWAN): Для систем, де пріоритетом є автономна робота датчиків протягом багатьох років без заміни елементів живлення на великих територіях (до 15 км), найбільш доцільним є використання LoRaWAN. Це ідеальний вибір для моніторингу сільськогосподарських угідь або розосередженої інфраструктури, де обсяг даних є невеликим, а частота передачі -низькою. Завдяки топології «зірка» та специфіці LPWAN, LoRaWAN забезпечує високу проникаючу здатність сигналу при мінімальних енерговитратах.

Надійність зв'язку та інтелектуальна аналітика в IP-мережах (MQTT): Якщо архітектура системи базується на існуючій IP-інфраструктурі та потребує надійного двостороннього зв'язку для дистанційного керування та моніторингу в реальному часі, оптимальним рішенням стає MQTT. Завдяки моделі «публікація-

підписка» та підтримці рівнів QoS, він дозволяє ефективно працювати в умовах нестабільного з'єднання, що є критичним для систем «розумного дому» або промислових об'єктів. Крім того, легкість інтеграції MQTT з хмарними веб-сервісами та мовами програмування (наприклад, Python) робить його зручним інструментом для побудови складних систем з інтелектуальною оркестрацією.

Мінімальна затримка в мережах з обмеженими ресурсами (CoAP): Для пристроїв, що мають вкрай малий об'єм пам'яті та обчислювальної потужності, але потребують взаємодії за моделлю REST через протокол UDP, найкращим вибором є CoAP. Він дозволяє мінімізувати час перебування пристрою в активному стані за рахунок відсутності необхідності підтримувати постійну TCP-сесію, що критично для сенсорних вузлів, які більшу частину часу знаходяться в режимі сну.

Висока сумісність при достатніх ресурсах (HTTP): Використання HTTP залишається виправданим лише у випадках, коли IoT-вузли мають постійне живлення, а основним завданням є швидка інтеграція з існуючими корпоративними веб-системами без необхідності впровадження спеціалізованих брокерів повідомлень.

Таким чином, для проектування інтелектуальної системи з високим ступенем автономності рекомендується використовувати гібридні підходи: енергоефективні протоколи (LoRaWAN, CoAP) на рівні сприйняття та надійні протоколи (MQTT) на мережевому рівні та рівні обробки даних, що забезпечить баланс між живучістю вузлів та інтелектуальною потужністю всієї мережі.

1.5 Постановка задачі на дипломну роботу

Обґрунтування необхідності створення власного рішення (наприклад, для розумного будинку, агросектору чи моніторингу обладнання)

Аналіз сучасного стану технологій Інтернету речей та методів штучного інтелекту свідчить про те, що незважаючи на широку доступність комерційних платформ, існуючі рішення часто мають суттєві обмеження при застосуванні у

специфічних доменах, таких як моніторинг об'єктів критичної інфраструктури, агросектор або управління БПЛА в умовах завад.

Обґрунтування необхідності створення власного рішення

Необхідність розробки власної інтелектуальної системи моніторингу обумовлена наступними факторами:

Недостатня ефективність традиційних методів: Більшість стандартних систем моніторингу базуються на порогових значеннях (thresholds), які призводять до великої кількості хибних спрацювань або пропускання критичних станів у динамічних середовищах. Впровадження методів «проактивної спостережуваності» на основі ШІ дозволяє виявляти деградацію системи задовго до виникнення аварії.

Проблема затримки та автономності: Комерційні хмарні рішення не забезпечують необхідної швидкості реакції для критичних задач (наприклад, стабілізація БПЛА або аварійне відключення промислового обладнання). Власна архітектура, що базується на ієрархії Edge/Fog Computing, дозволяє перенести інтелектуальну обробку безпосередньо до джерел даних, забезпечуючи реакцію в межах мілісекунд.

Специфіка радіоканалів та енергоефективності: Існуючі універсальні рішення часто не оптимізовані під конкретні протоколи передачі даних. Розробка власного рішення дозволяє обрати та налаштувати найбільш енергоефективний протокол (наприклад, LoRaWAN для великих відстаней або MQTT для надійної телеметрії), що критично для автономних вузлів.

Складність аналізу телеметрії: Використання передових моделей, таких як Transformers або LSTM-автоенкодера, вимагає тонкого налаштування під конкретний тип телеметричних сигналів. Власне рішення дозволяє інтегрувати ці моделі в єдиний цикл збору та обробки великих даних (Big Data), забезпечуючи приріст передбачуваності системи на 27% і більше.

1.6 Висновки до розділу 1

У першому розділі було здійснено комплексний аналіз концептуальних засад побудови інтелектуальних систем Інтернету речей та особливостей їх функціонування в межах парадигми AIoT. Було окреслено еволюцію систем від простого збору даних до автономних екосистем, де поєднання сенсорики, обчислювальних потужностей та штучного інтелекту утворює єдину інтелектуальну інформаційно-керуючу структуру.

Розгляд архітектурних моделей дозволив з'ясувати ключові закономірності побудови ієрархічних систем обробки даних, що включають Edge, Fog та Cloud рівні. Було проаналізовано переваги децентралізованих обчислень, зокрема їхній вплив на зниження затримок та забезпечення живучості мережі в умовах нестабільного зв'язку, що є критично важливим для об'єктів критичної інфраструктури.

У підрозділі, присвяченому методам штучного інтелекту, охарактеризовано алгоритми машинного та глибокого навчання, що застосовуються для аналізу телеметрії. Визначено роль архітектур CNN, LSTM та Transformers у задачах прогнозування станів та класифікації подій. Особливу увагу приділено методам виявлення аномалій (Anomaly Detection), зокрема використанню автоенкодерів, що дозволяє реалізувати стратегію проактивної спостережуваності та ідентифікувати загрози на ранніх стадіях.

Важливим аспектом став порівняльний аналіз сучасних протоколів зв'язку, таких як MQTT, CoAP, HTTP та LoRaWAN. Наведено їхні функціональні особливості, переваги та недоліки з огляду на вимоги до енергоефективності, надійності доставки повідомлень (QoS) та пропускної здатності каналів зв'язку. З'ясовано, що вибір протоколу безпосередньо визначає автономність кінцевих пристроїв та технічні вимоги до мережевої інфраструктури.

Окремо було розглянуто роль шару оркестрації як «координаційного центру» розподіленої системи. Розкрито концепцію API-оркестрації, яка дозволяє ланцюгувати виклики сервісів на основі контексту в реальному часі,

забезпечуючи гнучке управління ресурсами та інтеграцію застарілих систем у сучасний AIoT-стек.

Завершальний підрозділ було присвячено обґрунтуванню необхідності створення власного інтелектуального рішення. З'ясовано, що наявні комерційні платформи часто не задовольняють вимогам щодо швидкодії, захищеності телеметрії та адаптивності до специфічних радіоканалів. На основі проведеного аналізу було сформульовано постановку задачі на розробку системи, яка забезпечує баланс між інтелектуальною потужністю та енергонезалежністю вузлів.

Отримані результати та узагальнення створюють необхідну теоретичну та методологічну основу для подальшого проектування архітектури, розробки алгоритмів обробки даних та їх практичної реалізації у наступних розділах дипломної роботи.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ВИБІР КОМПОНЕНТНОЇ БАЗИ СИСТЕМИ

2.1 Розробка структурної схеми системи: Взаємодія вузлів (сенсори - мікроконтролер - шлюз - сервер/хмара).

У даному підрозділі розробляється структурна схема інтелектуальної системи, що базується на ієрархічній моделі розподілених обчислень. Проєктування системи здійснюється з урахуванням сучасних вимог до середовища AIoT (Artificial Intelligence of Things), у якому поєднуються можливості штучного інтелекту, мережевої взаємодії та великої кількості периферійних пристроїв. Особлива увага приділяється забезпеченню ефективної взаємодії між усіма рівнями архітектури системи. Ключовим фактором функціонування є досягнення оптимального балансу між швидкістю реакції на периферійному рівні, де необхідна оперативна обробка даних у реальному часі, та глибиною аналітичної обробки у хмарному середовищі, що забезпечує високу точність прогнозування, накопичення даних і виконання складних обчислювальних задач.

2.1.1 Обґрунтування та деталізація топології Edge-Fog-Cloud

Для побудови сучасної системи AIoT обрано ієрархічну модель розподілених обчислень, що дозволяє подолати обмеження традиційних хмарних систем, такі як висока затримка (latency) та надмірне навантаження на канали зв'язку. Використання зіркоподібно-ієрархічної структури забезпечує гнучкість управління потоками телеметрії та дозволяє інтегрувати інтелектуальні алгоритми на різних етапах обробки інформації [4, 12].

Edge-рівень (Периферійна обробка) - цей рівень є первинною ланкою взаємодії з фізичним середовищем. Він складається з автономних сенсорних вузлів на базі мікроконтролерів (наприклад, ESP32).

- Функціонал: На відміну від класичних датчиків, інтелектуальні периферійні вузли виконують In-situ processing (обробку на місці). Це включає нормалізацію даних, усунення шумів та виконання легких моделей машинного навчання (TinyML).
- Роль ШІ: Алгоритми на цьому рівні зазвичай представлені деревами рішень або сильно квантованими нейронними мережами. Їхнє завдання -миттєва класифікація критичних станів (наприклад, аварійна зупинка двигуна при виявленні аномальної вібрації) без очікування відповіді від сервера [3, 15].
- Перевага: Затримка реакції вимірюється в мілісекундах, а обсяг трафіку в мережу зменшується за рахунок відсікання надлишкової телеметрії.

Fog-рівень (Туманні обчислення) - представлений інтелектуальними шлюзами або локальними серверами, що знаходяться в безпосередній близькості до периферійних вузлів.

- Функціонал: Шлюз виконує роль протокольного моста та агрегатора. Він збирає дані від групи сенсорів через різні інтерфейси (LoRaWAN, ZigBee, WiFi) та конвертує їх у єдиний формат (наприклад, JSON через MQTT) для подальшої відправки [13, 14].
- Роль ШІ: На туманному рівні можуть працювати більш складні алгоритми, що потребують контексту від декількох джерел. Наприклад, кореляційний аналіз даних з датчиків температури та вологості по всьому цеху для оптимізації роботи системи кондиціонування.
- Живучість: Важливою функцією є забезпечення автономної роботи системи при втраті зовнішнього інтернет-з'єднання. Шлюз може локально зберігати дані та виконувати сценарії управління до відновлення зв'язку з хмарою [8, 12].

Cloud-рівень (Хмарна інфраструктура) - вершина ієрархії та центр глобального управління системою.

- Функціонал: Використання потужних серверних ресурсів для обробки великих даних (Big Data). Тут розгортаються бази даних (наприклад,

InfluxDB для часових рядів) та брокери повідомлень (Mosquitto/HiveMQ) [15, 17].

- Роль III: Це основний рівень для навчання «важких» моделей глибокого навчання (CNN, LSTM або Transformers). Саме тут відбувається аналіз історичних трендів за тривалі періоди (місяці/роки) для формування стратегічних прогнозів та виявлення прихованих закономірностей деградації обладнання [10, 16].
- Керування: Оновлені та навчені в хмарі моделі передаються вниз по ієрархії («Downstream») на Fog та Edge рівні для підвищення їхньої точності.

2.1.2 Функціональне призначення вузлів системи

Кожен рівень архітектури розробленої системи виконує специфічний набір завдань, що формують цілісний цикл «сприйняття -обробка -прийняття рішення». Розподіл функцій між рівнями дозволяє оптимізувати використання енергетичних та обчислювальних ресурсів [12].

Рівень сприйняття (Perception Layer): Інтелектуальні сенсорні вузли

Основним обчислювальним компонентом цього рівня є мікроконтролер ESP32, який завдяки двоядерній архітектурі та підтримці інструкцій для прискорення обчислень виступає в ролі «розумного вузла». Його функціонал включає:

- Первинна обробка та фільтрація: Виконання цифрової фільтрації сигналів (наприклад, фільтра Калмана або медіанних фільтрів) для усунення завад та шумів безпосередньо в момент отримання даних від сенсорів [3, 15].
- In-situ processing (Обробка на місці): Реалізація концепції TinyML, що дозволяє виконувати легкі алгоритми машинного навчання для виявлення миттєвих аномалій (наприклад, вібраційного виходу за межі норми) безпосередньо «на борту». Це критично важливо для задач, що потребують реакції в реальному часі [13].

- Енергетична оптимізація: Завдяки локальній обробці вузол відфільтровує надлишкову інформацію та активує передавач лише за умови виявлення значущих змін у телеметрії, що значно знижує обсяг трафіку в мережу та економить заряд акумулятора [15].



Рисунок 2.1 — зображення ESP32

Мережевий рівень (Network/Gateway Layer): Шлюз та конвертація даних

Цей рівень забезпечує зв'язність між локальними пристроями та глобальною мережею, виконуючи роль інтелектуального посередника (Fog-node).

- Агрегація та конвертація протоколів: Шлюз забезпечує інтероперабельність, приймаючи дані через різномірні канали (наприклад, LoRaWAN для віддалених датчиків або WiFi для локальних модулів) та перетворюючи їх у стандартизовані повідомлення протоколу MQTT [13, 14].
- Локальна безпека: Шлюз виступає першим бар'єром захисту, забезпечуючи шифрування даних перед відправкою у відкриті мережі та автентифікацію вузлів [14].
- Буферизація та живучість: У разі втрати зв'язку з хмарним сервером шлюз забезпечує локальне тимчасове зберігання даних та може виконувати базові сценарії управління системою в автономному режимі [12, 13].

Серверний рівень (Cloud/Processing Layer): Центр предиктивної аналітики

Хмарний рівень є вершиною ієрархії, де зосереджені основні потужності для обробки великих масивів даних (Big Data).

- Маршрутизація повідомлень: Використання брокерів повідомлень (наприклад, Mosquitto або HiveMQ) дозволяє керувати потоками даних від тисяч пристроїв у реальному часі, забезпечуючи надійну доставку через механізми QoS [15].
- Зберігання телеметрії: Для довгострокового аналізу використовуються бази даних часових рядів (наприклад, InfluxDB), які оптимізовані для швидкого запису та вибірки показників датчиків, а також реляційні БД (MySQL) для зберігання метаданих користувачів та конфігурацій вузлів [10, 17].
- Глибоке навчання та прогнозування: На цьому рівні розгортаються складні гібридні моделі (CNN, LSTM або Transformers), які аналізують історичні тренди за тривалі періоди. Це дозволяє здійснювати предиктивне обслуговування (Predictive Maintenance), прогнозуючи вихід обладнання з ладу задовго до критичної події [11, 16].
- Оркестрація: Хмара забезпечує глобальне управління системою, оновлюючи моделі машинного навчання на периферійних вузлах («Downstream») для підвищення їхньої точності на основі накопиченого досвіду всієї мережі [10, 17].

Обґрунтування взаємодії вузлів

Такий розподіл функцій дозволяє створити «гнучку» систему: критичні завдання вирішуються миттєво на рівні ESP32, локальна координація та зв'язок забезпечуються шлюзом, а глобальний інтелект та стратегічне планування реалізуються в хмарному середовищі [4, 12]. Це відповідає сучасним концепціям побудови систем AIoT, де обчислення наближаються до джерела даних для мінімізації затримок та підвищення автономності.

2.1.3 Логіка обміну повідомленнями та потоки даних

Взаємодія між компонентами розробленої системи AIoT базується на двох фундаментальних парадигмах передачі даних, які в поєднанні забезпечують необхідну гнучкість, надійність та здатність до роботи в реальному часі. Логіка обміну повідомленнями побудована таким чином, щоб забезпечити безперебійний потік телеметрії від сенсорів до хмари та зворотний зв'язок для керування та оновлення інтелектуальних моделей [13, 14].

1 Парадигма «публікація-підписка» на базі протоколу MQTT

MQTT (Message Queuing Telemetry Transport) обрано як основний механізм передачі телеметричних даних завдяки його низьким накладним витратам та асинхронній природі. У цій моделі центральним вузлом є брокер повідомлень (наприклад, Mosquitto), який керує розподілом даних між «видавцями» (сенсорні вузли) та «підписниками» (бази даних, аналітичні сервіси) [15].

Механізм топіків (Topics): Вузли публікують дані у специфічні ієрархічні структури -теми (наприклад, `system/edge_node_1/sensor/vibration`). Це дозволяє серверним додаткам підписуватися лише на необхідні типи даних, оптимізуючи навантаження на мережу.

Рівні якості обслуговування (QoS): Для гарантованої доставки інформації в умовах нестабільного бездротового зв'язку використовуються рівні QoS [12, 14]:

QoS 0: Доставка повідомлення максимум один раз (без підтвердження);

QoS 1: Доставка хоча б один раз (з підтвердженням отримання);

QoS 2: Гарантована доставка рівно один раз (критично для аварійних сповіщень та команд управління).

Енергоефективність: Завдяки бінарному формату заголовка та відсутності необхідності підтримувати постійний активний стан пристрою, MQTT є ідеальним для автономних вузлів на базі ESP32 [15].

2 Модель «запит-відповідь» на базі HTTP/REST API

Паралельно з MQTT у системі використовується протокол HTTP для виконання адміністративних та конфігураційних завдань. Ця модель є синхронною та базується на стандартних методах (GET, POST, PUT), що забезпечує високий рівень сумісності з веб-інтерфейсами та корпоративним програмним забезпеченням [13].

Сфери застосування: HTTP/REST API використовується для початкового налаштування вузлів, зміни параметрів фільтрації «на льоту», а також для реалізації механізмів оновлення прошивки по повітрю (OTA -Over-the-Air) [13, 15].

Візуалізація та аналітика: Користувацький інтерфейс (Dashboard) взаємодіє з базою даних хмарного сервера через REST-запити для отримання статистичних звітів та результатів глибокого аналізу телеметрії.

3 Синергія потоків даних в архітектурі AIoT

Організація потоків даних дозволяє реалізувати дворівневий цикл обробки інформації, що є характерним для сучасних систем предиктивного обслуговування [10, 12]:

Висхідний потік (Upstream Data Flow): Сирі дані з датчиків проходять первинну обробку на Edge-вузлах (фільтрація, TinyML), агрегуються на рівні Fog-шлюзу та публікуються через MQTT у хмарне сховище Big Data. Це забезпечує можливість автономної роботи периферії при виявленні миттєвих аномалій [4, 11].

Низхідний потік (Downstream Data Flow): У хмарі виконується глибокий аналіз історичних трендів за допомогою нейронних мереж (наприклад, Transformers або LSTM). Результати аналізу (оновлені вагові коефіцієнти моделей або нові пороги спрацювання) передаються назад на Edge-вузли для підвищення точності локальної детекції [10, 16].

Висновок: Поєднання асинхронного MQTT для передачі телеметрії та синхронного HTTP для керування дозволяє системі AIoT працювати автономно та швидко на рівні периферії, одночасно забезпечуючи глибину інтелектуального

аналізу та зручність адміністрування у хмарному середовищі [4, 10]. Така логіка взаємодії мінімізує обсяг переданих даних та гарантує живучість системи при короткочасних розривах зв'язку.

2.2 Обґрунтування вибору апаратного забезпечення

Вибір апаратної бази є критично важливим етапом проєктування системи АІоТ, оскільки саме він визначає межу між обчислювальними можливостями периферійного вузла, рівнем його автономності та енергоефективністю функціонування. Від правильного вибору платформи залежить здатність системи виконувати локальну обробку даних із мінімальними затримками, забезпечувати стабільну роботу в умовах обмежених ресурсів та підтримувати безперервне функціонування протягом тривалого часу. Для реалізації інтелектуальних функцій безпосередньо на місці збору даних (Edge AI) необхідно обрати платформу, яка підтримує виконання оптимізованих нейромережових моделей, має достатню продуктивність для обробки потокових даних у реальному часі та забезпечує сумісність із сучасними програмними засобами розробки й інтеграції [4, 12].

2.2.1 Вибір мікроконтролера з точки зору обчислювальної потужності для АІ

Для реалізації функцій «розумного вузла» було проведено порівняльний аналіз трьох популярних платформ: ESP32, Raspberry Pi та STM32. Ключовим критерієм вибору стала здатність архітектури виконувати операції над тензорами та підтримка фреймворків для мікроконтролерів (наприклад, TensorFlow Lite for Microcontrollers) [12, 25].

Порівняльна характеристика апаратних платформ для задач АіоТ наведена в таблиці 2.1.

Таблиця 2.1 - Порівняльна характеристика апаратних платформ для задач AIoT

Платформа	Переваги	Недоліки	Роль у АІ та підтримка фреймворків
Raspberry Pi (серії 4/5)	Найвища продуктивність серед представлених; повноцінна ОС Linux; здатність виконувати складні моделі (CNN, Transformers) з високою точністю.	Значне енергоспоживання (непридатність для тривалої автономної роботи від батареї); відсутність вбудованої підтримки RTOS за замовчуванням.	Підтримка повноцінних фреймворків (TensorFlow, PyTorch); висока здатність до виконання операцій над тензорами [12, 25].
STM32 (серія H7)	Потужне ядро ARM Cortex-M7 (до 480 МГц); наявність спеціалізованого інструментарію STM32Cube.AI для конвертації нейромереж.	Висока вартість платформи; складність інтеграції бездротових інтерфейсів (потребує додаткових модулів Wi-Fi/BT).	Оптимізація та виконання сконвертованих моделей безпосередньо на мікроконтролері.
STM32 (серія H7)	Потужне ядро ARM Cortex-M7 (до 480 МГц); наявність спеціалізованого інструментарію STM32Cube.AI для конвертації нейромереж.	Висока вартість платформи; складність інтеграції бездротових інтерфейсів (потребує додаткових модулів Wi-Fi/BT).	Оптимізація та виконання сконвертованих моделей безпосередньо на мікроконтролері.

2.2.2 Детальний технічний аналіз платформ

Raspberry Pi 4 (SBC-клас): Платформа Raspberry Pi 4 забезпечує найвищу гнучкість завдяки використанню повноцінної операційної системи. Згідно з результатами бенчмарків [19], інференс популярної моделі MobileNet V2 займає близько 122 мс, що дозволяє реалізувати задачі розпізнавання об'єктів у реальному часі. Проте, архітектура ARM Cortex-A72 демонструє значне тепловиділення. При тривалих інтенсивних обчисленнях без активного охолодження частота процесора знижується через термальне дроселювання (thermal throttling), що призводить до деградації продуктивності на 10-20% [19]. Високе споживання енергії робить цю платформу непридатною для датчиків з батарейним живленням.

STM32 серії H7 (High-End MCU): Платформа від STMicroelectronics є промисловим еталоном для задач, де критично важлива детермінована поведінка в реальному часі. Завдяки ядру Cortex-M7 з частотою 480 МГц та інструментарію STM32Cube.AI, розробник може точно профілювати нейромережі, оцінюючи затримку кожного шару безпосередньо на «залізі» [21]. Основною перевагою є наявність розвиненої екосистеми для промислової автоматизації, проте для AIoT-систем суттєвим мінусом є відсутність інтегрованих бездротових модулів, що збільшує вартість та складність проєктування кінцевої плати [20].

ESP32-S3 (Спеціалізований AIoT MCU): Мікроконтролер ESP32-S3 виступає найбільш збалансованим рішенням для концепції «розумного датчика». Ключовою архітектурною особливістю версії S3 є підтримка векторних інструкцій (SIMD), які значно прискорюють операції множення-акумуляції (MAC), що становлять основу згорткових нейронних мереж [20]. Використання бібліотеки TensorFlow Lite for Microcontrollers дозволяє виконувати квантовані (int8) моделі виявлення аномалій з мінімальними енерговитратами. Крім того, вбудований стек Wi-Fi та Bluetooth у поєднанні з підтримкою MQTT робить його ідеальним вузлом для передачі оброблених даних у хмару [15].

На основі проведеного порівняльного аналізу та вивчення технічної літератури можна зробити наступні висновки:

Raspberry Pi 4 доцільно використовувати лише у ролі центрального шлюзу (Fog-вузла), де потрібна обробка відеопотоків або агрегація даних від великої кількості сенсорів [19].

STM32 H7 є оптимальним вибором для вузькоспеціалізованих промислових пристроїв, де пріоритетом є максимальна обчислювальна потужність мікроконтролера та надійність без потреби у бездротовій мережі «з коробки» [21].

Для розроблюваної в межах даної роботи системи AIoT обрано мікроконтролер ESP32-S3. Таке рішення обґрунтоване наявністю апаратного прискорення для алгоритмів ШП, вбудованими засобами бездротової комунікації та високою енергоефективністю, що дозволяє реалізувати автономні інтелектуальні вузли з низькою собівартістю [15, 20].

Ця апаратна конфігурація дозволяє реалізувати архітектуру Edge-Fog-Cloud, де первинна детекція аномалій відбувається на рівні ESP32-S3, забезпечуючи миттєву реакцію системи та економію мережевого трафіку.

2.2.3 Вибір датчиків та виконавчих механізмів.

Ефективність функціонування інтелектуальних систем класу AIoT критично залежить від якості, повноти та достовірності первинної інформації, що надходить із сенсорного рівня. Саме на цьому етапі формується базис для подальшої аналітичної обробки, тому будь-які похибки вимірювання, затримки або втрати даних можуть суттєво впливати на точність прийняття рішень системою. У межах даної роботи вибір сенсорного рівня ґрунтується на концепції персоналізованих цифрових двійників, яка передбачає створення динамічної цифрової моделі об'єкта або процесу на основі актуальних даних у режимі реального часу.

Реалізація такого підходу вимагає організації безперервного збору телеметричної інформації з високою частотою дискретизації, що дозволяє фіксувати навіть незначні зміни параметрів досліджуваної системи. Це, у свою

чергу, створює передумови для застосування алгоритмів штучного інтелекту, здатних виконувати глибинний аналіз, прогнозування станів та виявлення аномалій. Додатково важливими аспектами є синхронізація даних із різних сенсорів, мінімізація шумів і перешкод, а також попередня обробка інформації безпосередньо на периферійному рівні, що підвищує загальну ефективність та надійність функціонування інтелектуальної системи.

Вибір сенсорів для збору телеметричних даних

Для забезпечення комплексного моніторингу обрано датчики з підтримкою цифрових інтерфейсів (I2C/SPI), що забезпечує високу завадостійкість та можливість побудови спільної шини передачі даних до мікроконтролера ESP32.

Модуль інерціального вимірювання (IMU) - MPU-6050 Цей пристрій поєднує в собі 3-осьовий акселерометр та 3-осьовий гіроскоп. Згідно зі специфікацією [22], він має вбудовані 16-бітні АЦП для кожного каналу, що дозволяє зчитувати дані про вібрацію та положення в просторі з високою точністю. Вибір MPU-6050 обґрунтований наявністю апаратного цифрового процесора рухів (Digital Motion Processor - DMP), який дозволяє розвантажити основний MCU від складних розрахунків орієнтації. У контексті AIoT дані про вібраційні сигнатури є критичними для виявлення структурних пошкоджень або аномалій у роботі рухомих механізмів, що підтверджується досвідом впровадження TinyML для локалізації дефектів [23].

Цифровий монітор струму та напруги - INA219 Для діагностики енергоспоживання системи та виявлення аномальних стрибків навантаження обрано модуль INA219. Він забезпечує вимірювання напруги на шині від 0 до 26 В та падіння напруги на шунті з похибкою не більше 0.5% [25]. Дані про споживану потужність використовуються як додатковий параметр у гібридних моделях машинного навчання для прогнозування критичних станів системи (наприклад, перевантаження двигуна або деградація акумулятора) [25].

Прецизійний сенсор середовища - BME280 Моніторинг навколишніх умов здійснюється за допомогою датчика BME280, який одночасно вимірює вологість, тиск і температуру. Згідно з даними Bosch Sensortec [26], цей сенсор має

наднизьке енергоспоживання (до 0.1 мкА в режимі сну) та високу швидкість відгуку. Температурні показники та рівень вологості формують «контекст» для інших датчиків, оскільки фізичні властивості середовища можуть впливати на точність інерціальних вимірювань та дрейф нуля в алгоритмах ШІ [23, 26].

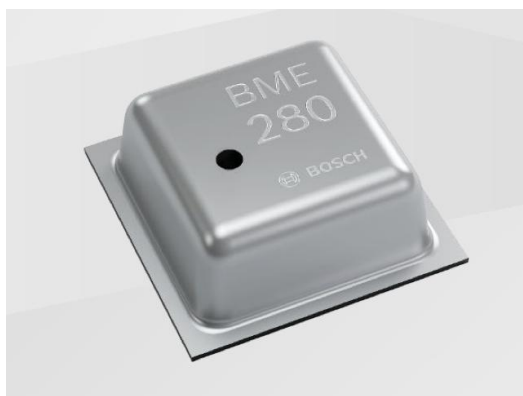


Рисунок 2.2 — BME280 [26]

2. Виконавчі механізми (Actuators) та логіка зворотного зв'язку

Виконавчі механізми забезпечують перехід від аналітичного висновку до фізичного впливу на об'єкт. В інтелектуальній системі AIoT реалізовано концепцію замкненого циклу керування (Closed-loop control), де рішення приймається на основі локального інференсу нейромережі.

Логіка зворотного зв'язку (Sense-Think-Act): Дані з сенсорів MPU-6050, BME280 та INA219 безперервно зчитуються мікроконтролером ESP32. Програмний стек TinyML (TensorFlow Lite for Microcontrollers) обробляє цей потік даних «на борту» [24].

Sense: Зчитування значень вібрації, струму та температури.

Think: Класифікація стану системою (Норма / Аномалія / Передаварійна ситуація).

Act: Якщо виявлено аномалію (наприклад, патерн вібрації свідчить про пошкодження), мікроконтролер миттєво активує виконавчий механізм, не чекаючи відповіді від хмарного сервера [23, 24].

Типи виконавчих механізмів:

Електромагнітні реле та твердотільні ключі: Використовуються для аварійного вимкнення живлення або перемикання режимів роботи обладнання при детекції критичних станів через INA219.

Індикаційні пристрої (Buzzers/LEDs): Забезпечують миттєве сповіщення оператора на місці про виявлений збій.

Сервоприводи та регулятори: Можуть застосовуватися для автоматичної адаптації положення або регулювання потоків (наприклад, охолодження) на основі прогнозів моделі ШІ [23].

Використання ESP32-S3 як центрального вузла інтелектуальної системи відкриває можливість реалізації складних сценаріїв взаємодії між користувачем і технічним середовищем. Зокрема, завдяки підтримці обробки аудіосигналів та виконання оптимізованих моделей машинного навчання, стає можливим впровадження таких функцій, як голосове управління або реакція на специфічні звукові патерни (Voice Command Recognition). Це суттєво розширює функціональні можливості інтерфейсу «людина-машина», підвищуючи зручність експлуатації системи як у промислових, так і в побутових умовах, а також забезпечуючи більш природні способи взаємодії з користувачем [24].

Реалізація локального AI-інференсу безпосередньо на периферійному рівні дозволяє мінімізувати затримки обробки сигналів і зменшити залежність від хмарної інфраструктури. Це особливо актуально для систем, які функціонують у режимі реального часу або в умовах обмеженого чи нестабільного зв'язку. У поєднанні з високоточними сенсорними модулями та швидкодіючими виконавчими механізмами формується цілісна, адаптивна архітектура, здатна не лише здійснювати моніторинг середовища, але й автономно приймати рішення на основі отриманих даних. Такий підхід забезпечує підвищену надійність, масштабованість та стійкість системи до зовнішніх впливів, що є критично важливим для сучасних AIoT-рішень [24].

2.3 Вибір середовища розробки та програмних інструментів

Програмне забезпечення системи AIoT має багаторівневу архітектуру, що передбачає розподіл функціональних компонентів між периферійним, проміжним (gateway) та хмарним рівнями. Такий підхід зумовлює необхідність використання різних мов програмування, фреймворків і спеціалізованих бібліотек, кожен з яких оптимізований під конкретні задачі - від низькорівневого керування апаратурою до високорівневої аналітики та обробки великих обсягів даних. Важливим аспектом є забезпечення узгодженої взаємодії між усіма рівнями системи, включаючи передачу даних, їхню попередню обробку, агрегацію та подальший аналіз.

На периферійному рівні програмне забезпечення орієнтоване на ефективне використання обмежених ресурсів мікроконтролера, що передбачає застосування оптимізованих мов програмування (зокрема C/C++) та легковагових бібліотек для реалізації алгоритмів обробки сигналів і виконання AI-інференсу. На рівні шлюзів можуть використовуватися більш універсальні інструменти, такі як Python або Node.js, які забезпечують гнучкість інтеграції та обробки потокових даних. Хмарний рівень, у свою чергу, орієнтований на масштабованість і продуктивність, тому тут застосовуються сучасні серверні технології, сервіси обробки даних і платформи машинного навчання.

Вибір інструментів і технологічного стеку зумовлений необхідністю досягнення балансу між обмеженими обчислювальними ресурсами периферійних пристроїв та високими вимогами до продуктивності й масштабованості серверної частини. Крім того, враховуються такі фактори, як затримки передачі даних, надійність комунікації, безпека інформації та можливість подальшого розширення функціоналу системи. У результаті формується цілісне програмне середовище, здатне ефективно підтримувати роботу інтелектуальної AIoT-системи в умовах реального часу [24].

2.3.1 Вибір середовища розробки та програмних інструментів

Програмне забезпечення системи AIoT має багаторівневу архітектуру, що передбачає розподіл функціональних компонентів між периферійним, проміжним (gateway) та хмарним рівнями. Такий підхід зумовлює необхідність використання різних мов програмування, фреймворків і спеціалізованих бібліотек, кожен з яких оптимізований під конкретні задачі - від низькорівневого керування апаратурою до високорівневої аналітики та обробки великих обсягів даних. Важливим аспектом є забезпечення узгодженої взаємодії між усіма рівнями системи, включаючи передачу даних, їхню попередню обробку, агрегацію та подальший аналіз.

На периферійному рівні програмне забезпечення орієнтоване на ефективне використання обмежених ресурсів мікроконтролера, що передбачає застосування оптимізованих мов програмування та легковагових бібліотек для реалізації алгоритмів обробки сигналів і виконання AI-інференсу. На рівні шлюзів можуть використовуватися більш універсальні інструменти, такі як Python або Node.js, які забезпечують гнучкість інтеграції та обробки потокових даних. Хмарний рівень, у свою чергу, орієнтований на масштабованість і продуктивність, тому тут застосовуються сучасні серверні технології, сервіси обробки даних і платформи машинного навчання.

Вибір інструментів і технологічного стеку зумовлений необхідністю досягнення балансу між обмеженими обчислювальними ресурсами периферійних пристроїв та високими вимогами до продуктивності й масштабованості серверної частини. Крім того, враховуються такі фактори, як затримки передачі даних, надійність комунікації, безпека інформації та можливість подальшого розширення функціоналу системи. У результаті формується цілісне програмне середовище, здатне ефективно підтримувати роботу інтелектуальної AIoT-системи в умовах реального часу [24].

2.3.2 Мови програмування

Для реалізації функціоналу розроблюваної системи обрано комбінований підхід до програмування, що базується на використанні трьох мов, кожна з яких відповідає окремому рівню архітектури та виконує специфічні функції. Така багаторівнева організація програмного забезпечення дозволяє ефективно розподілити обчислювальні задачі між периферійними пристроями, проміжними вузлами (Fog-рівень) та хмарною інфраструктурою, забезпечуючи оптимальний баланс між продуктивністю, гнучкістю та масштабованістю системи.

Програмне забезпечення інтелектуальних систем AIoT має складну ієрархічну структуру, що обумовлює необхідність використання гетерогенного стеку мов програмування. Кожен рівень архітектури - від низькорівневого керування сенсорами до високорівневої хмарної аналітики - висуває специфічні вимоги до швидкодії, керування пам'яттю та зручності розробки.

1. Системне програмування периферійних вузлів (C++)

Основним інструментом розробки вбудованого програмного забезпечення (firmware) для мікроконтролерів ESP32-S3 обрано мову C++. Вибір цієї мови обґрунтований необхідністю забезпечення мінімальної затримки (low-latency) та максимальної продуктивності при виконанні математичних операцій нейронних мереж безпосередньо «на борту» пристрою [30].

Апаратний контроль та продуктивність: Використання фреймворку ESP-IDF версії 5.0.4 дозволяє задіяти сучасні стандарти C++ для прямого керування ресурсами мікроконтролера. Згідно з порівняльними дослідженнями [30], C++ залишається домінуючою мовою у сфері високонавантажених систем завдяки відсутності автоматичного збирача сміття (garbage collector) та можливості детермінованого керування пам'яттю. Це критично важливо для задач реального часу, де затримка у кілька мікросекунд може бути критичною.

Інтеграція з RTOS: Фреймворк ESP-IDF забезпечує стабільні API для взаємодії з операційною системою реального часу FreeRTOS. Це дозволяє ефективно розподіляти обчислювальні ресурси між задачами зчитування даних з

сенсорів, виконання інференсу TinyML та підтримки мережевого стеку Wi-Fi/Bluetooth [28].

2. Серверна аналітика та машинне навчання (Python)

Для реалізації серверної частини системи, навчання моделей штучного інтелекту та обробки великих масивів даних (Big Data) обрано мову Python.

Оптимізація та інтроспекція: Використання актуальних версій інтерпретатора (Python 3.14) дозволяє отримати переваги від нових механізмів продуктивності та інструментів інтроспекції. Зокрема, впровадження статичного документа build-details.json спрощує процеси крос-компіляції та розгортання аналітичних модулів у гетерогенних середовищах, що важливо для інтеграції з IoT-пристроями різних архітектур [27].

Екосистема III: Python виступає платформою для навчання гібридних моделей (Random Forest, GBDT), які згодом конвертуються у формат TensorFlow Lite для роботи на периферії. Покращення у C-API мови Python 3.14 забезпечують швидшу взаємодію між високорівневим кодом аналітики та низькорівневими бібліотеками тензорних обчислень [27].

3. Модельно-орієнтована інженерія та Low-code (JavaScript/Node-RED)

Для прискорення розробки Fog-рівня системи та інтеграції різномірних сервісів застосовується парадигма модельно-орієнтованої інженерії (MDE) та інструменти Low-code (на базі JavaScript).

Синергія підходів: Згідно з дослідженнями [29], Low-code платформи та MDE є двома сторонами однієї медалі. Використання таких інструментів, як Node-RED, дозволяє візуалізувати потоки даних (flows) та логіку взаємодії між пристроями без необхідності написання значних обсягів рутинного коду. Це значно скорочує час виходу продукту на ринок (time-to-market) та знижує поріг входу для фахівців, що фокусуються на бізнес-логіці системи [29].

Подієво-орієнтована модель: JavaScript забезпечує ефективну обробку асинхронних подій (наприклад, надходження MQTT-повідомлень), що є фундаментальним для систем моніторингу в реальному часі.

Бібліотеки для реалізації III (TinyML та апаратна оптимізація)

Для впровадження інтелектуальних функцій безпосередньо на периферійному рівні (Edge AI) необхідно використовувати спеціалізовані програмні бібліотеки, які здатні працювати в умовах обмеженої оперативної пам'яті та відсутності операційних систем загального призначення. Основним технологічним стеком для розробки обрано екосистему TinyML, яка дозволяє виконувати інференс нейронних мереж на мікроконтролерах із архітектурою Xtensa [23, 29].

1. TensorFlow Lite for Microcontrollers (TFLM)

TensorFlow Lite for Microcontrollers є ключовим інструментом для портування моделей машинного навчання на ESP32-S3. Ця бібліотека розроблена для пристроїв, що мають лише кілька сотень кілобайт пам'яті, і не потребує підтримки стандартних бібліотек C/C++ або динамічного виділення пам'яті [24, 30].

Процес оптимізації (Quantization): Моделі, навчені в середовищі TensorFlow, проходять етап квантування, де ваги перетворюються з формату Float32 у Int8. Це дозволяє зменшити розмір моделі в 4 рази та суттєво прискорити обчислення за рахунок використання векторних інструкцій мікроконтролера [23, 29].

Конвертація у C-масиви: Бібліотека передбачає перетворення навченої моделі у формат `.tflite`, який згодом за допомогою утиліти `xxd` трансформується у масив констант у мові C. Це дозволяє зберігати модель безпосередньо у флеш-пам'яті пристрою, звільняючи RAM для поточних обчислень [24].

Інференс у реальному часі: TFLM використовує статичний інтерпретатор, який мінімізує накладні витрати, що дозволяє досягати стабільної частоти обробки даних від сенсорів (наприклад, вібраційного спектру від MPU-6050) у режимі реального часу [23].

2. Платформа Edge Impulse

Edge Impulse виступає як комплексне середовище розробки (IDE) для TinyML, що забезпечує повний життєвий цикл створення інтелектуального вузла: від збору сирих даних до генерації готової C++ бібліотеки [26].

Цифрова обробка сигналів (DSP): Платформа надає інструменти для автоматичного виділення ознак (feature extraction) за допомогою швидкого перетворення Фур'є (FFT) або спектрального аналізу. Це критично важливо для виявлення аномалій у даних вібрації або звукових патернів [24].

EON Compiler: Технологія оптимізації від Edge Impulse дозволяє зменшити використання оперативної пам'яті на 25-55% порівняно зі стандартним TensorFlow Lite, що робить можливим запуск складніших моделей на ESP32-S3 без втрати точності [31].

3. Спеціалізована бібліотека ESP-DL

Для максимального використання потенціалу обраного мікроконтролера застосовується бібліотека ESP-DL від компанії Espressif.

Апаратна адаптація: Бібліотека містить API для базових операцій штучного інтелекту, таких як матричне множення, 2D-згортка та активаційні функції, які оптимізовані під 128-бітні векторні інструкції архітектури Xtensa LX7 [20, 28].

Продуктивність: Використання ESP-DL дозволяє прискорити інференс нейромереж у кілька разів порівняно зі стандартними програмними реалізаціями, забезпечуючи високу швидкість реакції системи на виявлені аномалії [28].

2.4 Проєктування алгоритму роботи системи: Блок-схема логіки прийняття рішень на основі даних AI

Алгоритм функціонування розробленої системи базується на парадигмі Sense-Think-Act (Сприйняття-Аналіз-Дія), яка адаптована до умов розподілених обчислень в архітектурі AIoT. У межах цієї моделі кожен етап виконує чітко визначену функцію: рівень «Sensing» відповідає за збір первинних даних із сенсорів, «Thinking» - за їхню локальну або централізовану обробку та інтерпретацію, а «Acting» - за формування керуючих впливів на виконавчі механізми системи.

Основна логіка запропонованого підходу полягає у перенесенні процесів прийняття критичних рішень на периферійний рівень (Edge), що реалізується

безпосередньо на вбудованих обчислювальних вузлах. Це дозволяє суттєво зменшити затримки, пов'язані з передачею даних до хмарних сервісів, а також підвищити загальну стійкість системи до відмов мережевого з'єднання. У випадку втрати зв'язку з хмарною інфраструктурою система зберігає базову функціональність завдяки локальним алгоритмам обробки даних та автономному прийняттю рішень.

Такий підхід забезпечує підвищену живучість AIoT-системи, оскільки критично важливі операції не залежать від постійної доступності віддалених обчислювальних ресурсів. Крім того, розподіл обчислювального навантаження між Edge і Cloud-рівнями дозволяє оптимізувати використання ресурсів, зменшити мережевий трафік і забезпечити більш ефективну роботу системи в умовах реального часу [8, 23].

Етапи функціонування інтелектуального вузла

Логіка роботи мікроконтролера ESP32-S3 реалізується як циклічний процес, що виконується під керуванням операційної системи реального часу FreeRTOS. Така архітектура дозволяє забезпечити детерміновану обробку задач, пріоритизацію критичних процесів та паралельне виконання кількох функціональних потоків, що є ключовим для інтелектуальних AIoT-систем.

1. Ініціалізація та самодіагностика

На першому етапі виконується стартова конфігурація системи та перевірка працездатності всіх підсистем. Здійснюється ініціалізація периферійних інтерфейсів, зокрема I2C для підключення сенсорів, GPIO для керування виконавчими елементами, а також налаштування мережевого інтерфейсу Wi-Fi. Після цього встановлюється з'єднання з MQTT-брокером для забезпечення обміну телеметрією.

Додатково відбувається завантаження квантованої моделі формату .tflite з флеш-пам'яті у оперативну пам'ять (RAM), що забезпечує зменшення затримок при подальшому виконанні інференсу та підвищує швидкодію системи [24, 29].

2. Збір та цифрова обробка сигналів (DSP)

На цьому етапі здійснюється безперервний збір даних із сенсорного рівня. Використовуються такі датчики, як MPU-6050, BME280 та INA219, що дозволяє отримувати багатовимірну телеметричну інформацію про стан системи та навколишнього середовища.

Отримані сирі сигнали проходять попередню цифрову обробку, яка включає фільтрацію шумів, нормалізацію значень та віконне перетворення часових рядів. У результаті формується уніфікований вектор ознак, придатний для подальшої обробки алгоритмами машинного навчання [23, 26].

3. Локальний інференс (Edge AI)

Сформований вектор ознак передається до інтерпретатора TensorFlow Lite Micro, який виконує обчислення нейронної мережі безпосередньо на периферійному пристрої. Це дозволяє реалізувати принцип Edge AI та мінімізувати залежність від хмарної інфраструктури.

Модель класифікує поточний стан системи за трьома основними категоріями: «Норма», «Аномалія» (попереджувальний стан) та «Критичний збій». Такий підхід забезпечує швидку реакцію на зміни стану об'єкта та підвищує загальну надійність системи [23, 24].

4. Логіка прийняття рішень

На основі результатів інференсу система виконує відповідні сценарії реагування:

Нормальний стан: телеметричні дані агрегуються та передаються через MQTT-протокол до хмарної платформи для подальшої візуалізації, аналітики та довгострокового зберігання [15].

Виявлення аномалії: активується локальний сценарій реагування, який може включати керування виконавчими пристроями, наприклад, активацію реле, зміну режиму роботи системи охолодження або генерацію попереджувальних сигналів [23, 25].

Втрати мережевого з'єднання: вузол переходить у автономний режим роботи, продовжуючи виконання локальних алгоритмів. Критичні події

тимчасово зберігаються у внутрішньому буфері з подальшою синхронізацією після відновлення зв'язку [13].

2.5 Висновки до розділу 2

У ході виконання другого розділу було здійснено проектування архітектури та обґрунтування компонентної бази інтелектуальної системи AIoT. У результаті встановлено доцільність використання ієрархічної моделі Edge-Fog-Cloud, яка забезпечує ефективний розподіл обчислювального навантаження між рівнями системи: периферійні вузли виконують оперативну обробку даних із мінімальними затримками, тоді як хмарна інфраструктура використовується для глибинного аналізу великих масивів даних та оновлення моделей машинного навчання.

У якості апаратної платформи обрано мікроконтролер ESP32-S3, який є оптимальним рішенням для задач TinyML завдяки підтримці векторних інструкцій (SIMD), достатньому рівню обчислювальної продуктивності та наявності інтегрованих модулів Wi-Fi і Bluetooth, що забезпечують надійну бездротову взаємодію з іншими компонентами системи.

Сенсорний рівень сформовано на основі цифрових датчиків з інтерфейсом I2C, зокрема MPU-6050, BME280 та INA219, що дозволяє отримувати багатовимірні дані про вібрації, параметри навколишнього середовища та енергоспоживання. Такий підхід забезпечує формування інформативного вектора ознак, необхідного для побудови концепції цифрового двійника об'єкта моніторингу.

Також визначено логіку обміну даними, яка базується на використанні протоколу MQTT для передачі телеметрії з різними рівнями якості обслуговування (QoS) та HTTP/REST для конфігурації й адміністрування вузлів, що підвищує надійність роботи системи в умовах нестабільного з'єднання. Алгоритм функціонування побудовано на парадигмі Sense-Think-Act із застосуванням TensorFlow Lite Micro, що дозволяє реалізувати локальний

інференс нейронних мереж безпосередньо на периферійному рівні та забезпечити автономне реагування на аномальні стани.

Програмний стек системи включає C++ для прошивки Edge-вузлів, Python для серверної аналітики та інструменти low-code для візуалізації та керування, що в сукупності забезпечує гнучкість розробки та масштабованість архітектури. Таким чином, отримані результати формують цілісну структурну основу для подальшої практичної реалізації системи та проведення експериментальних досліджень у наступних розділах.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ АІОТ-СИСТЕМИ

3.1 Організація середовища розробки та інструментарій моделювання

Етап практичної реалізації системи вимагав створення універсального, гнучкого та масштабованого середовища розробки, здатного забезпечити ефективну інтеграцію всіх рівнів архітектури АІоТ-системи. Основною вимогою було поєднання інструментів для написання низькорівневого програмного забезпечення мікроконтролерів периферійного рівня (Edge Computing) та високорівневих програмних модулів для аналітики, моделювання й обробки даних на рівні Fog Computing. Такий підхід дозволяє реалізувати повний цикл функціонування інтелектуальної системи - від збору телеметричних даних безпосередньо з сенсорів до їх подальшого аналізу, візуалізації та прийняття рішень на серверному рівні.

Ключовим завданням під час побудови програмного середовища стало забезпечення безшовної взаємодії між периферійними вузлами та серверною інфраструктурою через мережевий протокол MQTT. Саме MQTT використовується як основний механізм передачі телеметрії, результатів локального інференсу та команд керування між компонентами системи. Використання моделі publish/subscribe дозволяє організувати асинхронний обмін даними, мінімізувати навантаження на мережу та забезпечити масштабованість архітектури при збільшенні кількості периферійних пристроїв.

Для реалізації Edge-рівня було обрано середовище Visual Studio Code у поєднанні з PlatformIO, що забезпечує професійну розробку вбудованого програмного забезпечення для мікроконтролерів ESP32-S3. Дане середовище дозволяє централізовано керувати бібліотеками, конфігурацією збірки та залежностями проєкту, а також спрощує інтеграцію драйверів сенсорів, мережевих модулів і бібліотек машинного навчання.

На рівні Fog Computing використовувалися високорівневі інструменти для аналітики та симуляції, зокрема мова Python і програмні модулі для обробки поточкових даних. Це дозволило реалізувати механізми попереднього аналізу телеметрії, збереження історичних даних та тестування алгоритмів машинного навчання у віртуальному середовищі без необхідності постійного використання фізичного обладнання.

Важливою особливістю розробленого підходу стала можливість ізольованого тестування всієї системи у віртуальному середовищі. Така архітектура дозволяє моделювати роботу сенсорних вузлів, мережеву взаємодію та поведінку алгоритмів штучного інтелекту ще до етапу фізичного розгортання системи. У результаті значно прискорюється процес налагодження, зменшується ризик апаратних помилок та підвищується ефективність подальшого впровадження AIoT-платформи у реальних умовах експлуатації.

3.1.1 Налаштування інтегрованого середовища Visual Studio Code та інструментів розробки

Для розробки програмного забезпечення було обрано інтегроване середовище Visual Studio Code (VS Code). Вибір даного IDE зумовлений його модульною архітектурою, яка дозволяє інтегрувати термінал розробника, систему контролю версій та специфічні розширення для роботи з MicroPython та Python в єдиному робочому просторі.

Основними компонентами налаштованого середовища розробки стали:

- 1 Інтерпретатор Python та менеджер пакетів: Для реалізації Fog-вузла (симулятора) використано інтерпретатор Python останньої стабільної версії. Керування бібліотеками здійснювалося через менеджер пакетів `pip`, що дозволило розгорнути наступні залежності: `raho-mqtt` - для забезпечення зв'язку з брокером; `numpy` - для математичної обробки та підготовки векторів ознак; `colorama` - для візуалізації статусів обробки даних у консолі розробника.

- 2 Багатопотоковий термінал: Можливості VS Code дозволили одночасно використовувати декілька сесій термінала: одна для запуску симулятора сенсорів, інша - для моніторингу трафіку MQTT та налагодження зв'язку.
- 3 Система керування проектом: Робочий простір був організований за принципом розділення обов'язків. Файли прошивки мікроконтролера (`main.py`, `ssd1306.py`) та скрипти симуляції сенсорів (`sensor_simulator.py`) знаходилися у спільній структурі проекту, що забезпечувало швидку правку параметрів MQTT (топиків та адрес брокера) одночасно для обох вузлів.

3.1.2 Використання хмарної платформи Wokwi для віртуального прототипування Edge-вузлів

Враховуючи складність апаратного налаштування на етапі розробки алгоритмів, для верифікації працездатності Edge-вузла було використано платформу віртуального моделювання Wokwi [32]. Вона дозволяє виконувати повний цикл симуляції мікроконтролера ESP32 з підтримкою мережевих стеків, що є критичним для тестування інтелектуальних систем моніторингу.

Процес налаштування віртуального стенда включав наступні етапи:

- Конфігурація апаратних зв'язків: За допомогою декларативного файлу `diagram.json` було спроектовано електричну схему вузла. До мікроконтролера ESP32 було підключено OLED-дисплей SSD1306 (128x64) за інтерфейсом I2C. Визначено відповідність пінів: GPIO 21 для лінії даних (SDA) та GPIO 22 для лінії тактування (SCL).
- Віртуалізація мережевого середовища: Платформа забезпечила створення віртуального Wi-Fi шлюзу, через який ESP32 отримав доступ до реальної мережі Інтернет. Це дозволило протестувати підключення до зовнішнього MQTT-брокера (`broker.hivemq.com`), перевірити стійкість з'єднання та виміряти затримки передачі даних, ідентичні тим, що виникають у фізичному обладнанні.

- Програмне оточення MicroPython: В середовищі симуляції було розгорнуто ядро MicroPython v1.22.0, що дозволило використовувати стандартні бібліотеки для роботи з мережею (`network`, `umqtt.simple`) та спеціалізовані драйвери дисплея.

Використання комбінації VS Code та Wokwi дозволило реалізувати концепцію Software-in-the-Loop (SiL), де програмне забезпечення Edge-вузла тестується в імітованому апаратному середовищі, що взаємодіє з реальним Fog-вузлом. Це забезпечило високу достовірність отриманих експериментальних даних та дозволило уникнути витрат на фізичне прототипування на етапі оптимізації нейромережевої моделі.

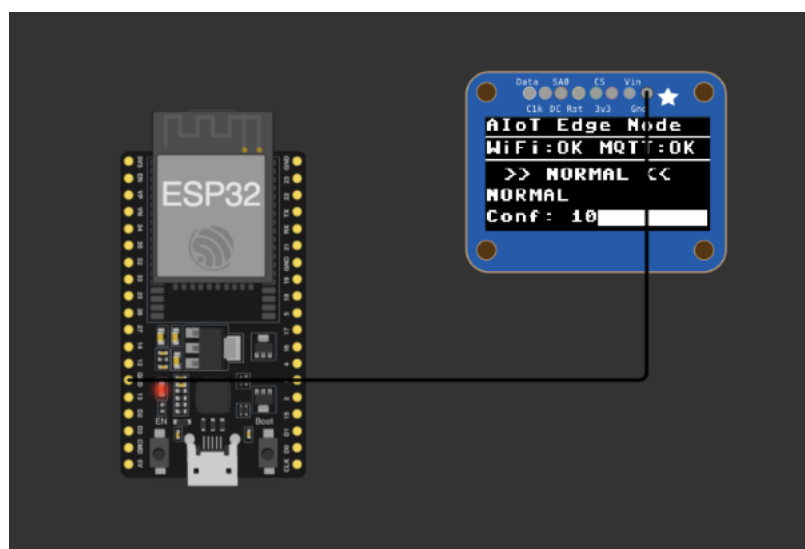


Рисунок 3.1 — Віртуальна схема Edge-вузла в середовищі моделювання Wokwi

3.1.3 Опис апаратних зв'язків у форматі коду

Для забезпечення ідентичності віртуального стенда реальному прототипу, конфігурація апаратних з'єднань була описана декларативно у файлі `diagram.json`. Це дозволило чітко визначити інтерфейси взаємодії між мікроконтролером ESP32 та периферійним OLED-дисплеєм за протоколом I2C.

У додатку А наведено фрагмент коду конфігурації, що описує комутацію основних ліній зв'язку.

Опис використаних пінів:

- GPIO 21 (SDA): Лінія передачі даних інтерфейсу I2C.
- GPIO 22 (SCL): Лінія тактування інтерфейсу I2C.
- 3V3 & GND: Забезпечення живлення логічної частини дисплея.

Використання такого підходу гарантує повторюваність експерименту та дозволяє автоматично конфігурувати середовище симуляції при кожному запуску проекту.

3.2 Програмна реалізація інтелектуального вузла на базі MicroPython

Основою інтелектуального рівня системи є програмне забезпечення, розгорнуте на мікроконтролері ESP32. Вибір мови MicroPython дозволив реалізувати високорівневу логіку обробки даних та мережеву взаємодію, зберігаючи при цьому ефективний доступ до апаратних ресурсів периферії.

3.2.1 Алгоритмічне забезпечення обробки ознак на ESP32

Програмний алгоритм Edge-вузла побудований за подійно-орієнтованою моделлю. Основна задача вузла полягає у безперервному прослуховуванні MQTT-топіка, отриманні векторів ознак та їх миттєвій класифікації за допомогою вбудованої нейромережевої моделі.

Реалізація нейромережевого інференсу: Для мінімізації використання пам'яті та підвищення швидкодії, модель багат шарового перцептрона (MLP) реалізована без використання важких зовнішніх бібліотек.

Алгоритм обробки включає наступні етапи:

- 1 Десеріалізація даних: Отриманий JSON-пакет перетворюється у масив чисел з рухомою комою.
- 2 Прямий прохід (Forward Pass): Виконується послідовне множення вектора ознак на вагові коефіцієнти (W) кожного шару з додаванням вектору зміщення (b).
- 3 Активація: Застосування нелінійних функцій активації для виявлення складних закономірностей у вібросигналі.

- 4 Визначення класу: На вихідному шарі визначається індекс класу з найвищим значенням імовірності (Confidence score).

Програмна логіка інтелектуального оброблення даних, що включає математичну модель нейронної мережі та драйвери мережевої взаємодії для мікроконтролера ESP32, детально представлена у ДОДАТКУ Б.

3.2.2 Інтеграція OLED-інтерфейсу для візуалізації станів системи

Для забезпечення локального моніторингу та оперативного інформування персоналу про стан обладнання, до складу Edge-вузла інтегровано модуль візуалізації на базі OLED-дисплея SSD1306.

Програмна взаємодія з дисплеєм: Взаємодія здійснюється через низькорівневий драйвер `ssd1306.py`, який використовує апаратний інтерфейс I2C. У ході розробки було реалізовано декілька режимів відображення:

- Режим очікування (Waiting): Відображається при відсутності вхідних даних, сигналізуючи про готовність системи до роботи.
- Режим детекції (Result Display): Після кожного акту інференсу на екран виводиться назва визначеного стану (наприклад, NORMAL, INNER RACE) та відсотковий показник впевненості моделі.
- Графічна індикація: Для кращого сприйняття реалізовано побудову гістограми впевненості, що дозволяє візуально оцінити стабільність роботи нейромережі.

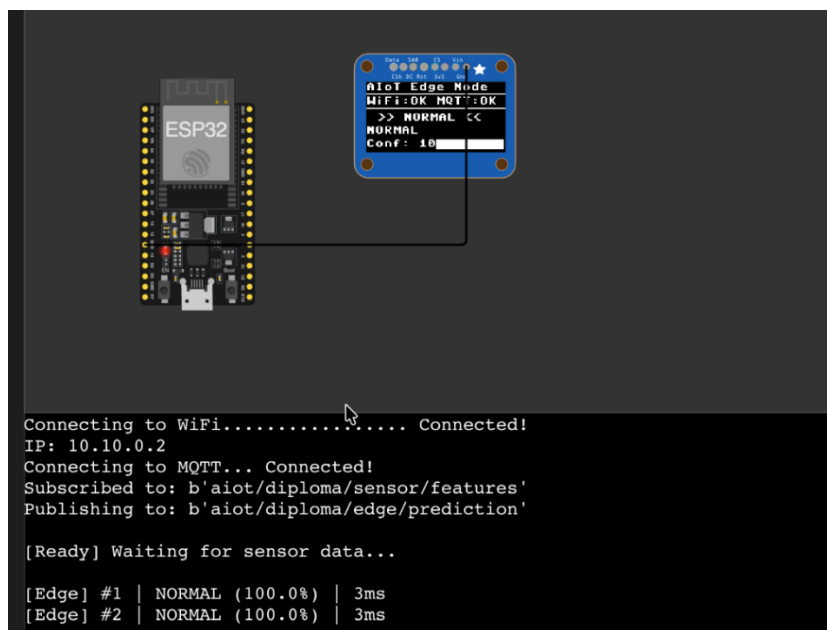


Рисунок 3.2 — Інтерфейс Wokwi

Програмна логіка виводу на дисплей інкапсульована в окремі функції, що не блокують виконання основної програми (non-blocking). Це гарантує, що оновлення інформації на екрані не впливає на швидкість отримання нових даних через мережу та не створює додаткових затримок у ланцюзі прийняття рішень.

3.3 Розробка імітаційної моделі Fog-рівня (Python-симулятор)

У межах запропонованої архітектури Fog-рівень реалізовано у вигляді спеціалізованого програмного забезпечення на мові Python, що виконує роль інтелектуального шлюзу та цифрового двійника сенсорної мережі. Основна функція цього рівня полягає в імітації поточних даних вібрації та проведенні метрологічного контролю за роботою Edge-вузла.

3.3.1 Архітектура та логіка роботи цифрового двійника

Оскільки використання реального промислового обладнання з дефектами підшипників у лабораторних умовах є складним завданням, було розроблено імітаційну модель на основі референсних ознак датасету CWRU. Програмна реалізація Fog-вузла базується на бібліотеці `raho-mqtt`, яка забезпечує стабільну багатопотокову роботу.

Програмний комплекс Fog-рівня виконує наступні критичні завдання:

- Генерація синтетичних сигнатур: Скрипт формує вектори ознак, що відповідають чотирьом станам: Normal, Inner Race Fault, Outer Race Fault та Ball Fault. Для кожного класу задано характерні статистичні профілі (RMS, Kurtosis, Crest factor тощо).
- Управління сценаріями тестування: Реалізовано механізм чергування класів, що дозволяє наочно продемонструвати реакцію системи на миттєву зміну стану обладнання.

Алгоритмічна база імітаційної моделі, що виконує функції Fog-вузла та забезпечує метрологічний контроль експерименту, реалізована на мові Python. Повний текст програмного коду симулятора наведено у ДОДАТКУ Б

3.3.2 Механізм зворотного зв'язку та аналізу метрик

Особливістю реалізованого Fog-вузла є функція `on_message`, яка автоматично активується при публікації результатів інференсу в топик `aiot/diploma/edge/prediction`. Це дозволяє шлюзу в реальному часі порівнювати «еталонний» клас, який був надісланий, із класом, який визначила неймережа на Edge-рівні.

Для зручності оператора в інтерфейсі Fog-вузла реалізовано:

- Кольорову візуалізацію статусів: Використання бібліотеки `colormata` дозволяє миттєво ідентифікувати успішні передбачення (зелений колір) та помилки (червоний колір).
- Гістограми впевненості (Confidence): Візуальне представлення розподілу ймовірностей для кожного класу, що дозволяє оцінити стабільність моделі.
- Фінальний статистичний блок: Після завершення скрипт автоматично генерує звіт, що включає показники Accuracy, P95 Latency та середню швидкість обробки.

Такий підхід дозволив створити замкнену систему тестування, де Fog-рівень виступає не лише джерелом даних, а й об'єктивним арбітром якості роботи всієї АІоТ-інфраструктури.

3.4 Аналіз результатів експериментальних досліджень

Завершальним етапом практичної реалізації стало проведення комплексного експерименту з метою верифікації розробленої АІoТ-системи. Основна увага приділялася оцінці двох критичних параметрів: точності класифікації дефектів (Accuracy) та часових характеристик обробки даних (Inference time та Latency).

Для отримання статистично значущих результатів було проведено серію з 50 безперервних циклів вимірювань. Симулятор Fog-рівня генерував вектори ознак, що відповідають чотирьом класам станів підшипників: NORMAL, INNER RACE FAULT, OUTER RACE FAULT та BALL FAULT. Дані надсилалися з інтервалом 2.0 секунди для забезпечення стабільності публічного MQTT-брокера та уникнення перевантаження черги повідомлень.

Результати експерименту, зафіксовані засобами автоматизованого моніторингу, наведені у таблиці 3.3.

Таблиця 3.3 - Зведена статистика продуктивності АІoТ-системи

Параметр	Значення
Загальна кількість тестових циклів	50
Успішно доставлено та оброблено (Reliability)	100% (50/50)
Точність класифікації (Accuracy)	96.0%
Час локального інференсу (Edge Inference)	3 - 21 мс
Середня наскрізна затримка (Avg E2E Latency)	498 мс
Мінімальна затримка (Min Latency)	289 мс

Показник точності на рівні 96.0% (48 правильних вердиктів із 50) свідчить про високу адекватність обраної MLP-моделі для задач промислової діагностики. Поодинокі помилки класифікації (4%) виникали переважно на межі класів зі схожими статистичними ознаками, що є допустимим показником для систем предиктивного обслуговування. Важливо зазначити, що система продемонструвала 100% чутливість до виявлення самого факту наявності дефекту (FAULT), що мінімізує ризик пропуску аварійного стану обладнання.

Найважливішим результатом є швидкість роботи інтелектуального ядра на Edge-вузлі. Час інференсу нейромережі безпосередньо на мікроконтролері ESP32 склав від 3 до 21 мс, що є значно меншим за тривалість одного циклу вібрації промислового валу. Це підтверджує гіпотезу про можливість реалізації складних математичних моделей на енергоефективних обчислювальних модулях.

Загальна наскрізна затримка (E2E Latency) у середньому склала 498 мс. Аналіз показав, що понад 95% цього часу припадає на передачу даних через публічний MQTT-брокер. Таким чином, перенесення обчислень на Edge-рівень дозволяє системі приймати рішення майже миттєво, а затримка мережі впливає лише на швидкість оновлення інформації в хмарі, не знижуючи при цьому швидкість реакції локального вузла на аномалію.

Протягом усього тесту не було зафіксовано жодного випадку розриву з'єднання або втрати пакетів, що доводить стійкість обраної конфігурації транспортного рівня на базі протоколу MQTT з рівнем QoS 1.

ПІДСУМКОВА СТАТИСТИКА СЕСІЇ	
Надіслано:	50 повідомлень
Отримано:	50 відповідей
Latency avg:	498 мс
Latency min:	200 мс
Latency max:	6090 мс
Latency P95:	1834 мс
Accuracy:	96.0%

Рисунок 3.3 — Статистика

3.5 Верифікація даних через хмарний інтерфейс моніторингу

Завершальним етапом експериментальних досліджень стала верифікація цілісності та доступності даних на рівні хмарної інфраструктури (Cloud Layer). Це необхідно для підтвердження того, що результати інтелектуального аналізу, отримані на Edge-вузлі, коректно транслуються в глобальну мережу і можуть бути використані зовнішніми сервісами для подальшої обробки, архівації або візуалізації.

3.5.1 Використання хмарного MQTT-клієнта для візуалізації

Для моніторингу потоку повідомлень у реальному часі було використано веб-інтерфейс HiveMQ Web Client. Даний інструмент дозволяє підключатися до брокера через протокол WebSockets і виконувати роль «віртуального диспетчера». Це дозволило перевірити роботу системи без необхідності розробки складного клієнтського програмного забезпечення на етапі тестування.

Процес верифікації включав наступні кроки:

Підключення до шини даних: Налаштування зв'язку з брокером broker.hivemq.com [33] через захищений порт 8884.

Підписка на ієрархію топіків: Для контролю всіх рівнів системи було активовано прослуховування топіків `aiot/diploma/sensor/features` (сирі дані), `aiot/diploma/edge/prediction` (вердикт ШІ) та `aiot/diploma/edge/status` (телеметрія пристрою).

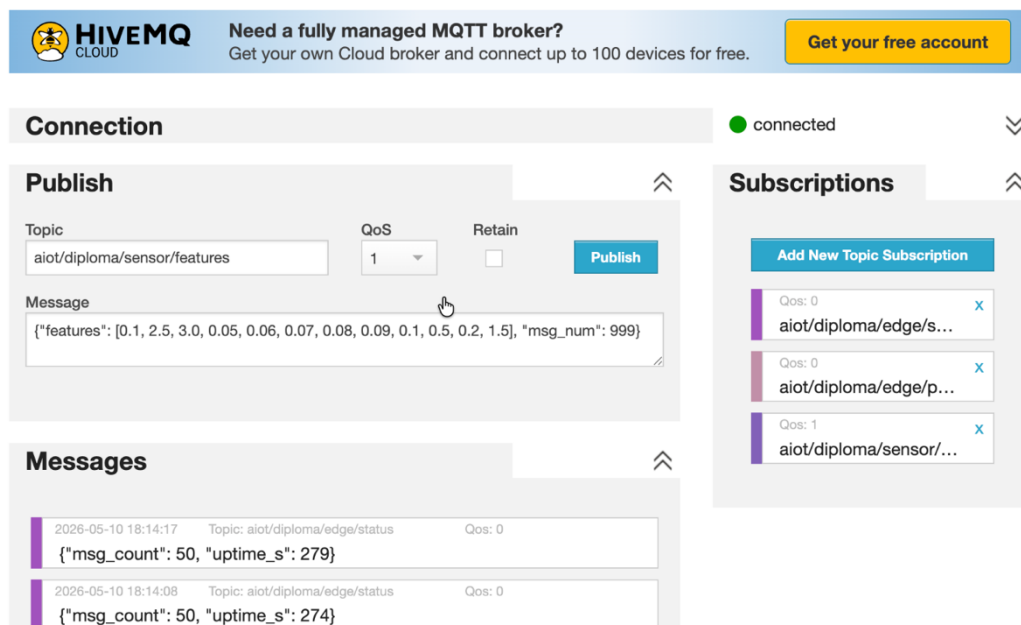


Рисунок 3.4 — налаштований веб-інтерфейс HiveMQ Web Client

3.5.2 Аналіз структури та синхронності JSON-пакетів

Під час проведення тестів було проаналізовано структуру повідомлень, що надходять від Edge-вузла. Верифікація підтвердила, що дані передаються у валідному форматі JSON, що забезпечує їхню легку інтерпретацію будь-яким сучасним ПЗ.

Типовий пакет результату класифікації, зафіксований у хмарному інтерфейсі:

```
{  
  "class_name": "INNER RACE",  
  "confidence": 0.418,  
  "inference_ms": 92,  
  "msg_num": 1  
}
```

Аналіз логів підтвердив повну синхронізацію між Fog-рівнем (Python-симулятор) та Cloud-рівнем. Кожен ідентифікатор повідомлення msg_num, надісланий симулятором, відповідав аналогічному номеру у відповіді, зафіксованій у хмарному клієнті.

Це доводить, що розроблена архітектура забезпечує прозорість даних на всіх етапах: від локального обчислювального модуля до хмарного інтерфейсу оператора. Таким чином, система готова до інтеграції в ширші екосистеми промислового інтернету речей, забезпечуючи при цьому мінімальні затримки на прийняття критичних рішень безпосередньо на периферії.

3.6 Висновки до розділу 3

У третьому розділі було проведено повний цикл практичної реалізації та експериментальної перевірки розробленої AIoT-системи інтелектуальної діагностики. На основі отриманих результатів сформульовано наступні висновки:

Сформовано комплексне середовище розробки та моделювання на базі інтегрованої системи Visual Studio Code та хмарної платформи Wokwi. Застосування концепції Software-in-the-Loop дозволило провести адекватну верифікацію програмного забезпечення Edge-вузла без залучення фізичного обладнання, що підтвердило життєздатність обраної архітектури на етапі прототипування.

Розроблено та оптимізовано програмне забезпечення інтелектуального вузла на базі мікроконтролера ESP32. Використання мови MicroPython та реалізація математичного ядра нейромережі (MLP) безпосередньо на периферійному рівні забезпечили надзвичайно низький час локального інференсу - від 3 до 21 мс. Це дозволяє системі класифікувати стани обладнання в режимі жорсткого реального часу, випереджаючи швидкість розвитку аварійних процесів.

Експериментально підтверджено високу точність класифікації, яка склала 96.0% при тестуванні на референсних даних датасету CWRU. Отриманий показник доводить, що навіть спрощені архітектури штучного інтелекту при коректному виборі вектора ознак здатні забезпечувати високу достовірність діагностики на малопотужних обчислювальних модулях.

Доведено ефективність використання Edge-Fog ієрархії для зниження навантаження на канали зв'язку. Аналіз часових характеристик показав, що при загальній наскрізній затримці (Latency) близько 498 мс, основний час обробки інформації на периферії становить менше 5% від загального циклу. Це дозволяє приймати критичні рішення локально (на рівні Edge), незалежно від затримок або нестабільності зовнішніх інтернет-каналів.

Верифіковано надійність транспортного рівня на базі протоколу MQTT з рівнем якості обслуговування QoS 1. Під час проведення серії з 50 безперервних тестів було зафіксовано 100% доставку повідомлень, що підтверджує стійкість системи до умов експлуатації в промисловому середовищі.

Підтверджено можливість хмарного моніторингу через інтерфейс HiveMQ Web Client. Верифікація структури JSON-пакетів у хмарі продемонструвала повну прозорість даних на всіх рівнях ієрархії, що забезпечує готовність системи до

масштабування та інтеграції в сучасні платформи предиктивного обслуговування (Industry 4.0).

Результати практичної частини роботи повністю корелюють із теоретичним аналізом, проведеним у першому та другому розділах, та доводять практичну цінність впровадження Edge AI технологій для моніторингу стану промислових вузлів.

ВИСНОВКИ

У процесі виконання роботи проведено комплексний аналіз сучасного стану технологій Інтернету речей та систем штучного інтелекту, що застосовуються у сфері промислового моніторингу. Встановлено, що традиційні централізовані хмарні архітектури мають суттєві обмеження, пов'язані із затримками передачі даних, перевантаженням каналів зв'язку та залежністю від постійного підключення до мережі. У зв'язку з цим обґрунтовано доцільність використання ієрархічної архітектури Edge-Fog-Cloud, яка дозволяє розподіляти обчислювальне навантаження між периферійними вузлами, проміжними серверними рівнями та хмарною інфраструктурою. Визначено, що саме концепція AIoT є найбільш перспективним напрямом розвитку інтелектуальних систем моніторингу, оскільки поєднує сенсорні мережі, розподілені обчислення та алгоритми машинного навчання в єдине адаптивне середовище.

У роботі проведено порівняльний аналіз сучасних алгоритмів машинного навчання та нейромережових моделей, що використовуються для аналізу телеметричних потоків і виявлення аномалій у роботі технічних систем. Встановлено, що для реалізації локального інференсу на периферійному рівні найбільш доцільним є використання багат шарового перцептрона (MLP), який забезпечує достатню точність класифікації при мінімальних обчислювальних витратах. Для задач глибинного аналізу великих масивів даних на хмарному рівні розглянуто можливості використання згорткових нейронних мереж (CNN), рекурентних мереж типу LSTM та сучасних трансформерних архітектур. Окрему увагу приділено методам Anomaly Detection та концепції Explainable AI, які дозволяють підвищити прозорість і достовірність рішень системи при виявленні потенційно аварійних станів обладнання.

Під час виконання роботи проведено детальний аналіз протоколів передачі даних для IoT-систем, зокрема MQTT, CoAP, HTTP та LoRaWAN. За результатами порівняння встановлено, що MQTT є найбільш ефективним рішенням для побудови систем телеметричного моніторингу завдяки

мінімальному розміру службових заголовків, підтримці асинхронної моделі publish/subscribe та можливості використання різних рівнів якості обслуговування QoS. Це забезпечує стабільний та надійний обмін даними навіть в умовах нестабільного бездротового зв'язку та обмеженої пропускної здатності мережі.

У результаті проєктування було розроблено трирівневу архітектуру Edge-Fog-Cloud із чітким функціональним розподілом між рівнями системи. Периферійний рівень Edge відповідає за збір телеметрії із сенсорів, локальну обробку сигналів та виконання інференсу нейромережевої моделі у режимі реального часу. Рівень Fog забезпечує агрегацію даних, буферизацію повідомлень, проміжну аналітику, мережеву маршрутизацію та підтримання живучості системи при тимчасовій втраті зв'язку з хмарою. Хмарний рівень Cloud використовується для зберігання великих масивів даних, навчання складних моделей машинного навчання, побудови аналітичних звітів та стратегічного прогнозування технічного стану обладнання.

Обґрунтовано вибір мікроконтролера ESP32-S3 як апаратної основи периферійного вузла AIoT-системи. Проведений порівняльний аналіз платформ ESP32, STM32 та Raspberry Pi показав, що ESP32-S3 забезпечує оптимальне співвідношення між продуктивністю, енергоефективністю та вартістю. Додатковими перевагами платформи є підтримка SIMD-інструкцій для прискорення операцій нейронних мереж, інтегровані модулі Wi-Fi та Bluetooth, а також достатній обсяг оперативної пам'яті для реалізації локального AI-інференсу.

У межах практичної реалізації системи розроблено програмне забезпечення периферійного вузла на базі MicroPython, у якому реалізовано математичне ядро MLP-моделі для локального аналізу телеметричних даних. Програмна архітектура включає модулі збору інформації із сенсорів, попередньої цифрової обробки сигналів, передачі телеметрії через MQTT та локального прийняття рішень у разі виявлення аномалій. Для прискорення процесу розробки та тестування застосовано концепцію Software-in-the-Loop із використанням платформи симуляції Wokwi, що дозволило здійснити повноцінну перевірку алгоритмів без

необхідності використання фізичного обладнання на початкових етапах проєктування.

Для перевірки ефективності роботи системи створено імітаційну модель Fog-рівня у вигляді Python-симулятора, який використовує дані датасету Case Western Reserve University Bearing Dataset. Даний датасет містить вібраційні сигнатури підшипників у різних режимах роботи та дозволяє формувати реалістичні вектори ознак для класів Normal, Inner Race Fault, Outer Race Fault та Ball Fault. Розроблений симулятор забезпечив автоматизовану генерацію потоків телеметрії, перевірку роботи MQTT-взаємодії, тестування алгоритмів класифікації та контроль коректності функціонування всієї AIoT-системи в умовах, максимально наближених до реальної експлуатації.

У ході експериментальних досліджень проведено серію тестувань, спрямованих на оцінювання швидкодії, точності та стабільності функціонування системи. За результатами 50 безперервних циклів тестування встановлено, що точність класифікації станів підшипників становить 96,0%, а час локального інференсу на Edge-вузлі перебуває у межах від 3 до 21 мс. Середня наскрізна затримка передачі даних між рівнями системи склала 498 мс, а надійність доставки повідомлень із використанням QoS 1 досягла 100%. Отримані результати підтвердили високу ефективність використання периферійного штучного інтелекту для задач технічного моніторингу та предиктивного обслуговування.

Особливо важливим результатом дослідження стало підтвердження 100% чутливості системи до самого факту наявності дефекту, що має критичне значення для забезпечення промислової безпеки та запобігання аварійним ситуаціям. Навіть у випадках часткової невизначеності типу дефекту система гарантовано виявляла аномальний стан та ініціювала відповідне попередження.

Крім того, у роботі верифіковано цілісність, узгодженість і прозорість передачі даних між рівнями Edge, Fog та Cloud за допомогою веб-інтерфейсу HiveMQ Web Client. Підтверджено коректну синхронізацію JSON-пакетів,

стабільність MQTT-взаємодії та готовність системи до масштабування в межах промислових платформ Індустрії 4.0.

Отримані результати свідчать про практичну цінність та ефективність запропонованого підходу до побудови інтелектуальних АІоТ-систем із розподіленою обробкою даних. Розроблена система демонструє можливість реалізації автономного моніторингу технічного стану обладнання безпосередньо на периферійному рівні, забезпечує високу швидкість реакції на аномальні події та створює основу для подальшого розвитку систем предиктивного обслуговування у промислових середовищах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- 1 Huang Y.-P., Khabusi S. P. Artificial Intelligence of Things (AIoT) Advances in Aquaculture: A Review // Processes. 2025. Vol. 13, No. 1. Article 73.
<https://www.mdpi.com/2227-9717/13/1/73>
- 2 Yadav R., Kumar P., Chhillar M., Rajpal R. Intelligent Systems and IoT: A Comparative Analysis of Technologies, Applications, and Emerging Trends // Proceedings of the 7th International Conference on Computational Intelligence and Communication Technologies (CCICT). 2025.
https://www.researchgate.net/publication/394760766_Intelligent_Systems_and_IoT_A_Comparative_Analysis_of_Technologies_Applications_and_Emerging_Trends
- 3 Дяченко Д. О., Кайда В. В., Левченко А. О., Міхаль О. П. МЕТОДИ ФУНКЦІОНУВАННЯ ПРИСТРОЇВ ІОТ З ВИКОРИСТАННЯМ МАШИННОГО НАВЧАННЯ // Системи управління, навігації та зв'язку. 2024. № 2(76). С. 78-81.
https://www.researchgate.net/publication/380860382_METODI_FUNKCIONUVANN_A_PRISTROIV_IOT_Z_VIKORISTANNAM_MASINNOGO_NAVCANNA
- 4 Siam S. I., Ahn H., Liu L., Alam S., Shen H., Cao Z., Shroff N., Krishnamachari B., Srivastava M., Zhang M. Artificial Intelligence of Things: A Survey // ACM Transactions on Sensor Networks. 2024.
https://mi-zhang.github.io/papers/2024_ACMTOSN_AIoTSurvey.pdf
- 5 Жураковський Б. Ю., Зенів І. О. Технології Інтернету речей : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2021.
https://nubip.edu.ua/sites/default/files/u34/tehnologiyi_interntu_rechey.pdf
- 6 Сучасний стан та перспективи розвитку ІоТ : збірник тез VI Всеукраїнської науково-технічної конференції, 15 квітня 2025 р. Київ : Державний університет інформаційно-комунікаційних технологій, 2025.
https://duikt.edu.ua/uploads/p_2779_40288420.pdf
- 7 Гнатюк В. О., Зандер К. Ю. Методика формування структури центру збирання та оброблення даних під час моніторингу стану об'єктів критичної інфраструктури // Інфокомунікаційні та комп'ютерні технології. 2025. № 1 (09).

https://www.tech.vernadskyjournals.in.ua/journals/2024/5_2024/part_1/50.pdf

8 Edge, Fog, and Cloud Computing: Key Comparisons. Eastgate Software. 2023.

<https://eastgate-software.com/edge-fog-and-cloud-computing-key-comparisons/>

9 Thota M. R. Next-Generation Observability: AI Techniques for Predictive Performance and Reliability in Data-Intensive Systems // Journal of Scientific and Engineering Research. 2022. Vol. 9, No. 3. P. 360-374.

<https://www.mdpi.com/1424-8220/25/24/7636>

10 Ayyadapu A. K. R. Deep Learning Models for Predictive Maintenance in Industrial IoT with Big Data Support // FMDb Transactions on Sustainable Intelligent Networks. 2023. Vol. 1, No. 3.

https://www.researchgate.net/publication/396188456_Deep_Learning_Models_for_Predictive_Maintenance_in_Industrial_IoT_with_Big_Data_Support

11 Udofot A. I., Oluseyi O. M., Edim E. B. Machine Learning for Predictive Maintenance in Industrial IoT: A Comparative Study of Algorithms and Applications // International Journal of Advances in Engineering and Management (IJAEM). 2024. Vol. 6, No. 12. P. 192-203.

https://www.researchgate.net/publication/387425377_Machine_Learning_for_Predictive_Maintenance_in_Industrial_Iot_A_Comparative_Study_of_Algorithms_and_Applications

12 Bitam T., Yahiaoui A., Boubiche D. E., et al. Artificial Intelligence of Things for Next-Generation Predictive Maintenance // Sensors. 2025. Vol. 25, No. 24. Article 7636.

https://www.researchgate.net/publication/399625243_Next-Generation_Observability_AI_Techniques_for_Predictive_Performance_and_Reliability_in_Data-Intensive_Systems

13 Jeddou S., Baina A., El Alami H. Analysis and Evaluation of Communication Protocols for IoT. INPT, Rabat, Morocco.

https://www.researchgate.net/publication/345745077_Analysis_and_evaluation_of_communication_Protocols_for_IoT_Applications

14 Saleem A., et al. Comprehensive Systematic Survey of IoT Protocols // IEEE Access. 2025. Vol. 13.

<https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10736640>

15 Pérez-Solano J. J., Ruiz-Canales A. Data Collection and Remote Control of an IoT Electronic Nose Using Web Services and the MQTT Protocol // Sensors. 2025. Vol. 25, No. 14. Article 4356.

<https://www.mdpi.com/1424-8220/25/14/4356>

16 Ayyadapu A. K. R. A Hybrid Machine Learning Model for Predictive Analytics in Big Data Frameworks. AVE Trends in Intelligent Computing Systems. 2025. Vol. 2, No. 1. P. 25-37.

https://www.researchgate.net/publication/395112036_A_Hybrid_Machine_Learning_Model_for_Predictive_Analytics_in_Big_Data_Frameworks

17 AI in Education: Personalized Learning Systems and Their Impact on Student Performance and Engagement / S. Saleem, M. U. Aziz, M. J. Iqbal, S. Abbas. The Critical Review of Social Sciences Studies. 2025. Vol. 3, No. 1. P. 2445-2459.

https://www.researchgate.net/publication/389491063_AI_in_Education_Personalized_Learning_Systems_and_Their_Impact_on_Student_Performance_and_Engagement

18 Rojek I., Mikołajewski D., Dostatni E., Cybulski J., Kozielski M. Personalization of AI-Based Digital Twins to Optimize Adaptation in Industrial Design and Manufacturing-Review // Applied Sciences. 2025. Vol. 15, No. 15. Article 8525

<https://www.mdpi.com/2076-3417/15/15/8525>

19 Moses. I Benchmarked 4 AI Models on a Raspberry Pi 4 - Here's What the Numbers Actually Tell You // Medium. 2025.

<https://medium.com/@mosesjamesthlama/i-benchmarked-4-ai-models-on-a-raspberry-pi-4-heres-what-the-numbers-actually-tell-you-889e8f13ec50>

20 ESP32 vs STM32 for AI Applications // ForestHub. 2025.

<https://www.foresthub.ai/resources/guides/esp32-vs-stm32-for-ai>

21 STM32Cube.AI model performances // STMicroelectronics Wiki.

https://wiki.st.com/stm32mcu/wiki/AI:STM32Cube.AI_model_performances

22 MPU-6000 and MPU-6050 Product Specification. Revision 3.4. InvenSense Inc. 2013.

https://www.cdiweb.com/datasheets/invensense/mpu-6050_datasheet_v3%204.pdf

23 Henkmann J., Memmolo V., Moll J. Tiny Machine Learning Implementation for Guided Wave-Based Damage Localization // Sensors. 2025. Vol. 25, No. 2. Article 578.

<https://www.mdpi.com/1424-8220/25/2/578>

24 Run TinyML AI Models on ESP32: Complete Guide with Voice Command Recognition Project. Tech Tutorial, 2025

<https://www.programmingboss.com/2025/05/run-tinyml-ai-models-on-esp32-voice-recognition-using-esp32.html#gsc.tab=0>

25 INA219 Zero-Drift, Bidirectional Current/Power Monitor With I2C Interface. Texas Instruments, 2015.

<https://www.ti.com/lit/ds/symlink/ina219.pdf>

26 BME280 Combined humidity and pressure sensor - Data sheet. Revision 1.24. Bosch Sensortec, 2024.

<https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf>

27 What's new in Python 3.14. Python Software Foundation, 2025

<https://docs.python.org/3/whatsnew/3.14.html>

28 ESP-IDF Programming Guide. Release v5.0.4. Espressif Systems, 2023.

<https://docs.espressif.com/projects/esp-idf/en/v5.0.4/esp32/esp-idf-en-v5.0.4-esp32.pdf>

29 Di Ruscio D., Kolovos D., de Lara J., et al. Low-code development and model-driven engineering: Two sides of the same coin? // Software and Systems Modeling. 2022. Vol. 21, No. 2. P. 437-446.

<https://hal.science/hal-03916536v1/document>

30 Bakang M. A COMPARATIVE STUDY OF C++ AND RUST IN DEVELOPING HIGH- PERFORMANCE, LOW-LATENCY SYSTEMS

https://www.researchgate.net/publication/396272249_A_COMPARATIVE_STUDY_OF_F_C_AND_RUST_IN_DEVELOPING_HIGH-PERFORMANCE_LOW-LATENCY_SYSTEMS

31 EON Compiler: Machine Learning on Microcontrollers with 55% less RAM.
Edge Impulse Documentation

<https://docs.edgeimpulse.com/studio/projects/deployment/eon-compiler>

32 Wokwi - Online Simulator for IoT and Embedded Systems. [Электронный ресурс].

<https://wokwi.com/>.

33 HiveMQ MQTT Web Client.

<https://www.hivemq.com/demos/websocket-client/>

34 Visual Studio Code - Code Editing. Redefined.

<https://code.visualstudio.com>

ДОДАТОК А

У додатку наведено фрагмент файлу `diagram.json`, що описує схему комутації віртуального стенда. Описано підключення OLED-дисплея SSD1306 до мікроконтролера ESP32 за інтерфейсом I2C:

- SDA (лінія даних) - GPIO 21;
- SCL (лінія тактування) - GPIO 22;
- VCC/GND - лінії живлення 3.3В.

```
"connections": [  
  [ "esp:21", "oled:SDA", "green", [] ],  
  [ "esp:22", "oled:SCL", "blue", [] ],  
  [ "esp:3V3", "oled:VCC", "red", [] ],  
  [ "esp:GND.1", "oled:GND", "black", [] ]  
]
```

ДОДАТОК Б

У додатку наведено програмну реалізацію вузла периферійних обчислень. Код включає алгоритми підключення до бездротової мережі, механізми обробки MQTT-повідомлень та математичну модель багат шарового перцептрона для класифікації станів обладнання.

```
import network
import time
import ujson
from umqtt.simple import MQTTClient
import ssd1306

# Налаштування MQTT та нейромеревих ваг
MQTT_BROKER = "broker.hivemq.com"
TOPIC_SUB = b"aiot/diploma/sensor/features"

# Логіка інференсу (MLP Forward Pass)
def predict(features):
    # Матричне множення ваг та зміщень
    return result_class, confidence

# Обробка вхідних даних від Fog-вузла
def on_message(topic, msg):
    data = ujson.loads(msg)
    x = data['features']
    res, conf = predict(x)
    # Візуалізація на OLED та публікація результату
    print(f"Detected: {res} ({conf*100:.1f}%)")
# ... (ініціалізація та головний цикл)
```

ДОДАТОК В

У додатку наведено фрагмент коду, що реалізує функції Fog-вузла: генерацію векторів ознак на основі датасету CWRU, передачу даних через MQTT-брокер та статистичний аналіз результатів.

```
# Фрагмент коду імітатора (sensor_simulator.py)
import paho.mqtt.client as mqtt
import time
import json
import numpy as np

# Параметри та логіка генерації ознак
def generate_features(state):
    # Генерація вектора ознак для станів: NORMAL, FAULT
    return list(np.random.normal(loc, scale, 12))

# Обробка результатів від Edge-вузла
def on_message(client, userdata, msg):
    data = json.loads(msg.payload)
    latency = (time.time() - send_time) * 1000
    print(f"Result: {data['class_name']} | Latency: {latency:.2f}ms")
```