КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

кібербезпеки та комп'ютерної інженерії

(назва випускової кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

на тему:

« Розробка алгоритму агрегації та попередньої обробки даних на периферійних пристроях IoT з метою зменшення затримок та навантаження на хмарну інфраструктуру»

Бондаренко Софія Олександрівна

(прізвище, ім'я та по батькові здобувача повністю)

Київ 2026 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

кібербезпеки та комп'ютерної інженерії

(назва кафедри)

ЗАТВЕРДЖУЮ

Завідувач кафедри

Делембовський М.М.

„___" _____20___року

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

«Розробка алгоритму агрегації та попередньої обробки даних на периферійних пристроях IoT з метою зменшення затримок та навантаження на хмарну інфраструктуру»

(назва)

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач Бондаренко Софія Олександрівна

(прізвище, ім'я та по батькові повністю)

123 «Комп'ютерна інженерія»

(спеціальність)

Комп'ютерні системи та мережі

(освітня програма)

Група КСМ-22

Керівник Шабала Є.Є.

(прізвище та ініціали)

доцент кафедри кібербезпеки та комп'ютерної інженерії, кандидат технічних наук

(вчене звання, науковий ступінь)

Рецензент_____

(прізвище та ініціали)

Ідентичність підтверджую

Київ 2026

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій
Випускова кафедра: кібербезпеки та комп'ютерної інженерії
Ступінь вищої освіти: бакалавр
Спеціальність: 123 «Комп'ютерна інженерія»
Освітня програма: Комп'ютерні системи та мережі

ЗАТВЕРДЖУЮ
Завідувач кафедри
Делембовський М.М.
„___" _____20___року

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

Бондаренко Софія Олександрівна
(прізвище, ім'я та по батькові здобувача)

1. Тема роботи «Розробка алгоритму агрегації та попередньої обробки даних на периферійних пристроях IoT з метою зменшення затримок та навантаження на хмарну інфраструктуру»

затверджена наказом ректора КНУБА № 577/52-15/13.2/26 від «14» травня 2026 року

2. Керівник роботи

доцент кафедри кібербезпеки та комп'ютерної інженерії, кандидат технічних наук Шабала Євгенія Євгенівна
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

3. Термін подання здобувачем роботи до захисту 30 травня 2026

4. Зміст пояснювальної записки за розділами:

Р. 1.Теоретичні основи та аналіз підходів до обробки даних у iot та edge computing

Р. 2. Постановка задачі та моделювання iot-середовища

Р. 3. Розробка алгоритму агрегації та попередньої обробки даних

Р. 4. Експериментальне дослідження та оцінювання ефективності розробленого алгоритму

5. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1	01.05.2026
Розділ 2	18.05.2026
Розділ 3	13.05.2026
Розділ 4	25.05.2026
Остаточне оформлення роботи	06.06.2026
Направлення роботи для перевірки на плагіат	
Попередній захист роботи	
Направлення роботи на рецензування	

6. Дата видачі завдання 10.02.2026

Керівник _____ _____

Здобувач _____ _____

РЕЗЮМЕ (SUMMARY) *до кваліфікаційної випускової роботи здобувача*	ПІБ *Бондаренко Софія Олександрівна* *Bondarenko Sofiia Oleksandrivna*		
ЗВО	Київський національний університет будівництва і архітектури		
Тема *(українською та англійською)*	«Розробка алгоритму агрегації та попередньої обробки даних на периферійних пристроях IoT з метою зменшення затримок та навантаження на хмарну інфраструктуру» «Development of an aggregation algorithm and data preprocessing on IoT peripheral devices to reduce latency and load on cloud infrastructure»		
Освітній ступінь	Бакалавр		
Факультет	Автоматизації і інформаціних технологій		
Випускова кафедра	Кібербезпеки та комп'ютерної інженерії		
Спеціальність	123 «Комп'ютерна інженерія»		
Освітня програма	Комп'ютерні системи та мережі		
Керівник	Шабала Євгенія Євгенівна		
Обсяг роботи:	*Поснювальна записка, стор.*	*Розділів*	*Презентація, кількість слайдів*
	83	4	12
Розділ 1	Теоретичні основи та аналіз підходів до обробки даних у iot та edge computing		
Розділ 2	Постановка задачі та моделювання iot-середовища		
Розділ 3	Розробка алгоритму агрегації та попередньої обробки даних		
Розділ 4	Експериментальне дослідження та оцінювання ефективності розробленого алгоритму		

Висновки по роботі	Розроблений алгоритм агрегації на основі ковзного вікна з адаптивним порогом забезпечує суттєве зменшення навантаження на хмарну інфраструктуру та контрольовану затримку передачі даних. Кількість пакетів, що передаються у хмару, скорочується у 10 разів для датчиків вологості, у 34 рази для температурних датчиків та до 45 разів для датчиків вібрації. Час обробки пакету на edge-вузлі становить 0,039 мс і не зростає при збільшенні кількості датчиків у 5 разів, що підтверджує масштабованість рішення. Аномалії передаються негайно, для решти даних верхня межа затримки — 60 с. Затримка каналу edge–хмара стабільна (~50,8 мс) в усіх сценаріях. Відсоток втрачених пакетів не перевищує 1,1% у штатному режимі та 2,0% у стресовому сценарії (50 датчиків, 20% аномалій).
Ключові слова:	IoT (Internet of Things), агрегація даних, попередня обробка даних, периферійний вузол (edge node), ковзне вікно, адаптивний поріг, виявлення аномалій, коефіцієнт стиснення, MQTT, затримка передачі, Python, pub/sub архітектура, Intel Lab Data.
Keywords:	IoT (Internet of Things), data aggregation, data preprocessing, edge node, sliding window, adaptive threshold, anomaly detection, compression ratio, MQTT, transmission latency, Python, pub/sub architecture, Intel Lab Data.

Здобувач Бондаренко С. О. _____/ _____

Керівник Шабала Є. Є. _____/ _____

АНОТАЦІЯ

Кваліфікаційна випускна робота присвячена розробленню алгоритму агрегації та попередньої обробки даних на периферійних пристроях IoT з метою зменшення затримок та навантаження на хмарну інфраструктуру.

Актуальність теми зумовлена стрімким зростанням кількості підключених IoT-пристроїв та пов'язаним із цим збільшенням обсягів даних, що передаються у хмару, що призводить до перевантаження каналів зв'язку та зростання операційних витрат на хмарну інфраструктуру. Розроблений алгоритм реалізує триступеневий механізм прийняття рішення про передачу на основі ковзного вікна з адаптивним порогом: виявлення точкових аномалій методом z-score, детектування зміни режиму роботи датчика та watchdog-таймер гарантованої доставки. Достовірність синтетичної моделі IoT-середовища підтверджена статистичною верифікацією за критерієм Колмогорова–Смирнова на основі реального набору даних Intel Lab Data. Ефективність алгоритму досліджена в трьох сценаріях навантаження: базовому (10 датчиків), середньому (30 датчиків) та стресовому (50 датчиків, 20% аномалій). Алгоритм забезпечує скорочення кількості пакетів, що передаються у хмару, у 10–45 разів залежно від типу датчика при часі обробки 0,039 мс, що не зростає зі збільшенням навантаження у 5 разів.

Ключові слова: IoT, edge computing, агрегація даних, ковзне вікно, адаптивний поріг, виявлення аномалій, потокова обробка даних, коефіцієнт стиснення, затримка передачі, хмарна інфраструктура, MQTT, Python.

ABSTRACT

The qualification graduation work is dedicated to the development of an algorithm for data aggregation and preprocessing on IoT edge devices in order to reduce latency and load on cloud infrastructure.

The relevance of the topic is determined by the rapid growth in the number of connected IoT devices and the associated increase in data volumes transmitted to the cloud, leading to communication channel overload and rising cloud infrastructure costs. The developed algorithm implements a three-stage transmission decision mechanism based on a sliding window with an adaptive threshold: point anomaly detection using z-score, sensor operating mode shift detection, and a watchdog timer for guaranteed delivery. The validity of the synthetic IoT environment model is confirmed by statistical verification using the Kolmogorov–Smirnov test against the real Intel Lab Data dataset. The algorithm's efficiency was evaluated across three load scenarios: baseline (10 sensors), medium (30 sensors), and stress (50 sensors, 20% anomalies). The algorithm reduces the number of packets transmitted to the cloud by 10–45 times depending on the sensor type, with a processing time of 0.039 ms that remains constant despite a 5-fold increase in load.

Keywords: IoT, edge computing, data aggregation, sliding window, adaptive threshold, anomaly detection, stream data processing, compression ratio, transmission latency, cloud infrastructure, MQTT, Python.

ЗМІСТ

ВСТУП

Розвиток технологій Інтернету речей (IoT) зумовив безпрецедентне зростання кількості підключених пристроїв та обсягів даних, що ними генеруються. За прогнозами аналітичних агентств, до 2030 року кількість IoT-пристроїв перевищить 30 мільярдів одиниць, а щодобовий обсяг даних, що передаватимуться від них, сягне декількох зетабайт. Традиційна хмаро-орієнтована архітектура, за якої необроблені дані з датчиків передаються безпосередньо до хмарної інфраструктури для обробки та зберігання, виявляється неефективною в умовах таких масштабів: виникають суттєві затримки передачі, перевантаження каналів зв'язку та надмірне навантаження на хмарні ресурси.

Парадигма edge computing (периферійних обчислень) пропонує альтернативний підхід, за якого частина обчислень переноситься безпосередньо на периферійні пристрої — граничні вузли, розташовані в безпосередній близькості до джерел даних. Попередня обробка та агрегація даних на периферії дозволяє суттєво скоротити обсяг переданих даних, знизити затримки реагування на критичні події та зменшити навантаження на хмарну інфраструктуру. Водночас задача ефективної агрегації є нетривіальною: алгоритм має забезпечувати збереження інформаційної цінності даних, своєчасне виявлення аномалій та гнучке управління рішенням про передачу даних у хмару.

Отже, розробка алгоритму агрегації та попередньої обробки даних на периферійних пристроях IoT є актуальною науково-прикладною задачею, вирішення якої має практичне значення для побудови ефективних IoT-систем у галузях промисловості, розумних міст, охорони здоров'я та інших.

Метою роботи є розробка та дослідження алгоритму агрегації та попередньої обробки даних на периферійних пристроях IoT, що забезпечує зменшення затримок і навантаження на хмарну інфраструктуру при збереженні достатньої повноти інформації.

Для досягнення поставленої мети визначено такі задачі:

– проаналізувати архітектурні підходи до побудови IoT-систем та методи обробки потокових даних на периферії;

– формалізувати задачу агрегації даних на edge-вузлі та визначити критерії ефективності алгоритму;

– розробити модель IoT-середовища, що включає моделі датчиків, локальних каналів зв'язку та каналу edge–хмара;

– розробити алгоритм агрегації на основі методу ковзного вікна з адаптивним порогом та детектуванням аномалій;

– реалізувати симуляційне середовище для дослідження алгоритму в різних умовах навантаження;

– верифікувати синтетичні дані симуляції на основі реального набору даних Intel Lab Data;

– провести експериментальне дослідження за трьома сценаріями навантаження та оцінити ефективність розробленого алгоритму.

Об'єкт дослідження — процес обробки та передачі потокових даних в IoT-системах із периферійними обчислювальними вузлами.

Предмет дослідження — алгоритм агрегації та попередньої обробки даних на периферійних IoT-пристроях, що мінімізує затримки та навантаження на канали зв'язку і хмарну інфраструктуру.

Методи дослідження: У роботі застосовано методи імітаційного моделювання для побудови та дослідження IoT-середовища; статистичні методи для верифікації синтетичних даних (критерій Колмогорова–Смірнова) та оцінювання параметрів розподілів; методи аналізу часових рядів для детектування аномалій; порівняльний аналіз для зіставлення ефективності різних підходів до обробки даних.

Наукова новизна одержаних результатів полягає у розробці алгоритму агрегації даних на периферійних IoT-вузлах, який поєднує метод ковзного вікна з адаптивним статистичним порогом виявлення аномалій, механізм сторожового таймера для гарантованої передачі та триєдину стратегію прийняття рішення про передачу (агреговані дані / аномалія / watchdog-пакет). Обґрунтовано критерій

стиснення CR як ключовий показник ефективності агрегації та досліджено його залежність від параметрів алгоритму і характеристик вхідного потоку.

Практичне значення одержаних результатів: Розроблені алгоритм та симуляційне середовище можуть бути використані при проектуванні та налаштуванні IoT-систем промислового моніторингу, систем розумних будівель і міської інфраструктури. Реалізований програмний модуль агрегації дозволяє відтворити результати дослідження та адаптувати параметри системи до конкретних умов застосування.

Структура та обсяг роботи: Дипломна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. У першому розділі розглянуто теоретичні основи IoT-архітектур, парадигму edge computing, методи потокової обробки та агрегації даних. У другому розділі формалізовано задачу дослідження, побудовано моделі IoT-середовища та потоків даних, описано симуляційне середовище. У третьому розділі представлено архітектуру та детальний опис розробленого алгоритму агрегації. У четвертому розділі наведено результати експериментального дослідження та порівняльний аналіз ефективності алгоритму.

РОЗДІЛ 1

ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ПІДХОДІВ ДО ОБРОБКИ ДАНИХ У IoT ТА EDGE COMPUTING

1.1 Особливості та архітектура IoT-середовищ

1.1.1 Поняття Інтернету речей

Інтернет речей (Internet of Things, IoT) у сучасному науково-технічному розумінні розглядається як глобальна інфраструктура інформаційного суспільства, що забезпечує надання розширених сервісів завдяки взаємозв’язку фізичних і віртуальних об’єктів на основі наявних та еволюціонуючих інформаційно-комунікаційних технологій [1]. Таке трактування підкреслює, що IoT не обмежується лише набором підключених датчиків, а являє собою цілісне середовище, у межах якого відбуваються збирання, передавання, обробка та використання даних для підтримки прикладних сервісів.

Ключовою рисою IoT є інтеграція цифрового та фізичного середовищ. Згідно з підходом NIST, IoT-пристрій зазвичай має щонайменше один перетворювач, тобто сенсор або виконавчий механізм, для безпосередньої взаємодії з фізичним світом, а також мережевий інтерфейс для обміну даними з цифровою інфраструктурою. Саме тому IoT-системи належать до класу кіберфізичних систем, у яких інформаційні процеси безпосередньо пов’язані зі спостереженням за станом об’єктів, параметрами середовища або впливом на них.

Важливою особливістю Інтернету речей є також його розподілений та масштабований характер. У межах одного IoT-середовища можуть одночасно функціонувати велика кількість пристроїв різного призначення, що генерують неоднорідні дані та взаємодіють через різні канали зв’язку [4]. Це зумовлює необхідність використання узгоджених архітектурних підходів, які забезпечують інтероперабельність компонентів, керованість інфраструктури та ефективну організацію потоків даних.

1.1.2 Рівні архітектури IoT (пристрої, мережа, обробка даних)

У науковій літературі та стандартах архітектура IoT подається у різних варіантах, однак у більшості випадків її можна звести до кількох базових функціональних рівнів. Зокрема, в рекомендації ITU-T Y.2060 наведено еталонну модель IoT, яка включає чотири шари: рівень пристроїв, мережевий рівень, рівень підтримки сервісів і застосунків та прикладний рівень [1]. Для цілей цієї роботи доцільно використовувати спрощене представлення, у якому виділяються три ключові рівні: рівень пристроїв, рівень мережевої взаємодії та рівень обробки даних, оскільки саме така модель найкраще відображає шлях руху даних від сенсора до сервісу аналітики або прийняття рішень.

Рівень пристроїв охоплює сенсори, актуатори, вбудовані контролери та інші кінцеві вузли, які забезпечують первинне зчитування параметрів середовища, локальне перетворення сигналів і, за потреби, виконання керувальних дій [2]. Саме на цьому рівні формується первинний потік даних, який у подальшому передається до шлюзів, периферійних вузлів або хмарних сервісів. Залежно від апаратних можливостей окремі пристрої можуть виконувати й елементарну локальну обробку, проте в більшості випадків їхні ресурси істотно обмежені.

Мережевий рівень забезпечує передавання даних між кінцевими пристроями, шлюзами, edge-вузлами та зовнішніми сервісами. Згідно з ITU-T Y.2060, на цьому рівні важливу роль відіграють не лише функції мережевої взаємодії, а й шлюзові можливості, зокрема протокольне перетворення між різними технологіями доступу [1]. Практично це означає, що IoT-система часто поєднує кілька типів з’єднань, наприклад ZigBee, Bluetooth, Wi-Fi, LTE або Ethernet, а шлюзи забезпечують сумісність між локальними мережами пристроїв і ширшою мережею передавання даних [2].

Рівень обробки даних включає локальні шлюзи, периферійні сервери, edge- або fog-вузли та хмарні платформи, на яких виконуються зберігання, фільтрація, агрегація, аналітика та візуалізація даних [3]. У традиційній централізованій моделі значна частина обчислень переноситься до хмари, однак зі зростанням обсягів телеметрії та вимог до швидкодії такий підхід спричиняє додаткові затримки і

збільшує навантаження на мережеву інфраструктуру. Саме тому сучасні IoT-архітектури дедалі частіше передбачають попередню обробку інформації ближче до джерела її виникнення.

1.1.3 Характеристики потоків даних IoT

Однією з визначальних ознак IoT-середовищ є те, що дані в них мають переважно потоковий характер. На відміну від пакетної обробки, де набір даних формується заздалегідь і надалі обробляється як завершений масив, у IoT інформація надходить безперервно або квазібезперервно, часто у часовому порядку та з вимогою оперативного реагування. Це зумовлює підвищені вимоги до швидкості обробки, своєчасності доставки повідомлень і мінімізації затримок між моментом виникнення події та її аналізом.

Ще однією важливою характеристикою IoT-потоків є їхня гетерогенність. Як зазначають Corral-Plaza та співавт., дані, що надходять від різних IoT-джерел, часто мають різну структуру, формат подання та змістове наповнення, через що системи змушені виконувати попередню нормалізацію та підготовку інформації перед її подальшим аналізом. У реальних системах це можуть бути часові ряди вимірювань, подієві повідомлення, логічні стани обладнання, службова телеметрія або метадані про самі пристрої, що значно ускладнює централізовану обробку без проміжних механізмів узгодження формату та зменшення обсягу даних.

Окрему проблему становить якість даних, отриманих від IoT-пристроїв. NIST підкреслює, що сенсорні дані, які відображають параметри фізичного світу, завжди пов’язані з певною невизначеністю. Крім цього, у практичних сценаріях потоки можуть містити шуми, пропуски, асинхронність надходження, дублікати або аномальні значення, що виникають через обмеження апаратного забезпечення, нестабільність каналу зв’язку чи вплив зовнішнього середовища. Тому попередня обробка даних, включаючи очищення, фільтрацію та нормалізацію, є необхідною передумовою для побудови надійних IoT-рішень.

1.1.4 Обмеження IoT-пристроїв (ресурси, енергоспоживання)

Суттєвою особливістю IoT-середовищ є обмеженість ресурсів значної частини кінцевих пристроїв. У RFC 7228 зазначено, що constrained-node networks формуються з вузлів, для яких характерні суттєві обмеження за енергоспоживанням, пам’яттю та обчислювальними ресурсами [2]. Це означає, що для багатьох IoT-пристроїв недоцільно або навіть неможливо використовувати повноцінні протокольні стеки, складні алгоритми аналізу чи ресурсоємні механізми захисту, характерні для традиційних серверних або персональних систем.

Для формалізації ступеня обмеженості RFC 7228 пропонує умовний поділ пристроїв на класи. Зокрема, пристрої класу 0 мають обсяг оперативної пам’яті значно менший за 10 KiB і кодову пам’ять значно меншу за 100 KiB, пристрої класу 1 — близько 10 KiB RAM і близько 100 KiB flash-пам’яті, а пристрої класу 2 — близько 50 KiB RAM і 250 KiB flash-пам’яті [2]. Для класу 0 характерна потреба у використанні проксі або шлюзів, оскільки такі вузли зазвичай не здатні безпосередньо та безпечно взаємодіяти з Інтернетом. Пристрої класів 1 і 2 мають ширші можливості, але навіть для них критичними залишаються ощадливе використання пам’яті, пропускної здатності каналу та енергії.

Крім суто апаратних обмежень, IoT-пристрої часто мають і функціональні обмеження щодо керування, моніторингу та безпеки. NIST вказує, що багато IoT-пристроїв не можуть бути доступні, керовані або контрольовані так само, як традиційні IT-системи; це ускладнює централізоване адміністрування, масштабне оновлення та моніторинг великої кількості вузлів. Також доступність, ефективність і повнота вбудованих засобів кібербезпеки в IoT часто є нижчими, ніж у звичайних обчислювальних системах, а деякі механізми захисту можуть створювати неприйнятні затримки або бути недосяжними через апаратні обмеження.

Таким чином, обмеженість IoT-пристроїв формує базову суперечність між потребою у безперервному зборі та передаванні даних і необхідністю економного використання обчислювальних, енергетичних і мережевих ресурсів [3-5]. Саме ця суперечність обґрунтовує доцільність перенесення частини функцій з агрегації,

фільтрації та попередньої обробки на проміжні периферійні вузли, які мають більші ресурси, ніж кінцеві сенсори, але розташовані ближче до джерела даних, ніж віддалена хмарна інфраструктура.

1.2 Парадигма edge computing

1.2.1 Поняття edge та fog computing

Стрімке зростання кількості IoT-пристроїв, збільшення інтенсивності потоків даних та підвищення вимог до швидкості реакції систем зумовили обмеженість виключно централізованої моделі обробки інформації. У традиційному хмарному підході значна частина обчислень виконується у віддалених дата-центрах, що є ефективним для масштабованого зберігання та обробки великих масивів даних, однак не завжди забезпечує прийнятну затримку для задач, чутливих до часу відгуку. У зв’язку з цим у сучасних розподілених системах сформувався перехід до моделей, у яких частина обчислювальних функцій переноситься ближче до місця виникнення даних, тобто до периферії мережі [3], [6].

У науковій літературі edge computing зазвичай розглядають як парадигму, що передбачає виконання обчислень, зберігання та попередньої обробки даних на вузлах, розміщених поблизу джерел цих даних. У праці W. Shi та співавт. підкреслено, що логіка edge computing полягає в тому, що обчислення мають виконуватися поблизу джерел даних, оскільки саме там це є найбільш ефективним з точки зору затримки, трафіку та конфіденційності [4]. M. Satyanarayanan розглядає edge computing як закономірний етап розвитку обчислювальних систем, пов’язаний із потребою забезпечення наднизької затримки та підтримки нових сервісів у 5G-мережах і кіберфізичних середовищах [5]. У документах ETSI ця ідея конкретизується через концепцію Multi-access Edge Computing, яка надає хмароподібні можливості та IT-середовище безпосередньо на краю мережі доступу, забезпечуючи високу пропускну здатність, контекстну обізнаність і малу затримку [6].

Поряд із поняттям edge computing у спеціалізованій літературі широко використовується термін fog computing. На відміну від edge-підходу, який часто асоціюють із обробкою безпосередньо на кінцевих або максимально наближених до них вузлах, fog computing зазвичай описують як багаторівневу проміжну інфраструктуру між кінцевими пристроями та хмарою. У моделі NIST fog-вузли визначаються як фізичні або віртуальні компоненти, тісно пов'язані з кінцевими пристроями або мережами доступу, які забезпечують обчислювальні ресурси, обмін даними та комунікаційні сервіси між периферійним рівнем і централізованою хмарою [7]. У документі OpenFog Reference Architecture також наголошується, що fog computing є доповненням і розширенням традиційної хмарної моделі, яке переміщує обчислення ближче до периферії та потенційно безпосередньо до сенсорів і актуаторів [8].

Попри близькість цих понять, edge і fog computing не є повністю тотожними. У джерелах NIST та OpenFog прямо зазначено, що fog computing нерідко помилково ототожнюють з edge computing, хоча між ними існують концептуальні відмінності [7-8]. Fog-підхід є більш ієрархічним, передбачає наявність проміжних вузлів, федерацію сервісів і розподілене керування ресурсами, тоді як edge computing у вузькому розумінні акцентує саме на виконанні обчислень у найближчій до джерела точці [7-10]. У межах цієї роботи доцільно розглядати edge computing як ширшу парадигму винесення обчислень на периферію, а fog computing — як один із способів організації такого середовища на проміжному, багаторівневому рівні між IoT-пристроями та хмарною інфраструктурою [9-10], [12].

1.2.2 Відмінності між edge, fog та cloud computing

Cloud computing є централізованою моделлю, у якій обчислювальні ресурси, сховища, додатки та сервіси надаються зі спільного пулу конфігурованих ресурсів

через мережу на вимогу користувача [3]. Основною перевагою хмари є практично необмежена масштабованість, еластичність та можливість централізованого адміністрування великих обсягів даних і сервісів. Однак саме централізований характер хмарної моделі стає її слабким місцем у сценаріях, де потрібні миттєва реакція, врахування локального контексту або мінімізація передаваного трафіку [3-4].

Головна відмінність між cloud, fog та edge computing полягає у місці виконання обчислень і зберігання даних. У хмарній моделі обробка зосереджується переважно у віддалених дата-центрах. В edge computing значна частина функцій переноситься безпосередньо до мережевої периферії — на локальні шлюзи, мікросервери, промислові контролери, базові станції або навіть кінцеві пристрої. У fog computing між кінцевими вузлами та хмарою формується проміжний розподілений шар із набору fog-вузлів, які можуть бути географічно розосередженими, але логічно об’єднаними в єдину систему [7-8], [11-12].

Відмінності між цими підходами чітко проявляються у характеристиках затримки, пропускної здатності та контекстної обізнаності. Обробка в хмарі потребує передавання даних до віддалених центрів обробки, що збільшує час відгуку та створює додаткове навантаження на транспортну мережу. Edge computing, навпаки, орієнтований на зменшення затримки, скорочення шляху даних і локальне врахування контексту. У документах ETSI MEC наголошується, що така модель забезпечує наднизьку затримку, високу пропускну здатність і доступ до мережевої інформації в реальному часі [6]. Fog computing також орієнтований на низьку затримку, але водночас підкреслює географічну розподіленість, підтримку мобільності, федерацію та взаємодію кількох проміжних вузлів [7].

Ще одна важлива відмінність стосується способу інтеграції з IoT-системами. У cloud-моделі IoT-пристрої переважно виконують роль джерел сирих даних, які передаються для подальшого аналізу у віддалену інфраструктуру. У fog- та edge-підходах між джерелом даних і хмарою з’являється розподілений контур

попередньої обробки, де можливе локальне фільтрування, агрегація, виявлення подій, кешування та прийняття первинних рішень. Саме тому сучасні дослідження дедалі частіше розглядають не жорстке протиставлення cloud та edge, а безперервний IoT–edge–fog–cloud continuum, у якому різні обчислювальні шари виконують різні функції залежно від вимог конкретного застосування [9-10], [12].

1.2.3 Переваги обробки даних на периферії

Ключовою перевагою edge computing є скорочення затримки обробки даних. Якщо інформація аналізується ближче до місця її виникнення, зменшується кількість проміжних передач через мережу та прискорюється реакція системи на події. Саме тому edge-підхід є особливо цінним у промислових IoT-системах, медичних сервісах, транспортних системах, відеоаналітиці, системах безпеки та інших застосуваннях, де критичними є часові характеристики [4-6], [9]. У низці українських праць також підкреслюється, що перенесення обробки на периферію дозволяє мінімізувати час прийняття рішень і зменшити навантаження на канали передавання даних [11-12].

Не менш важливою перевагою є зменшення мережевого трафіку та навантаження на хмарну інфраструктуру. У багатьох IoT-сценаріях значна частина даних має локальний або тимчасовий характер і не потребує повного передавання у хмару. Їх попереднє узагальнення, фільтрація або перетворення на локальному вузлі дозволяє передавати лише релевантні результати, а не повні сирі потоки. У роботах Shi та співавт. та Laroui та співавт. наголошується, що саме це дає змогу зменшити тиск на пропускну здатність мережі й зробити обробку даних економнішою [4], [9]. Подібні висновки містяться і в українських роботах, де edge-системи розглядаються як інструмент локалізації трафіку, зниження навантаження на центри обробки даних та оптимізації транспортного рівня [11-12].

Периферійна обробка також підвищує автономність і стійкість системи. Якщо частина функцій виконується локально, система здатна продовжувати роботу навіть за нестабільного або тимчасово недоступного з’єднання з хмарою.

Satyanarayanan відзначає, що cloudlet може у разі відмови віддаленої хмари тимчасово виконувати її критичні функції [5]. У сучасних edge-платформах, зокрема в описаних українськими дослідниками рішеннях на базі Raspberry Pi, локальних брокерів повідомлень та вбудованих ОС, акцент робиться саме на здатності працювати в умовах нестійкого зв'язку, забезпечуючи локальне зберігання, обробку і відновлення після збоїв.

Ще однією перевагою є покращення конфіденційності та контекстної обізнаності. Якщо сирі дані не покидають локальний сегмент мережі, а у хмару передаються лише агреговані результати або події, зменшуються ризики витоку чутливої інформації та спрощується дотримання нормативних вимог. Крім того, edge-вузли краще враховують локальний контекст: стан мережі, географічне розташування, доступність обладнання, час виникнення події та характеристики середовища. ETSI MEC прямо пов'язує edge computing із можливістю використання контекстної інформації мережі в реальному часі, а NIST і OpenFog наголошують на важливості контекстної обізнаності та географічної локалізації вузлів [6-8].

Для теми даної дипломної роботи особливе значення має те, що edge computing створює методологічну основу для реалізації алгоритмів агрегації та попередньої обробки даних безпосередньо на периферійних вузлах. Саме на цьому рівні доцільно виконувати первинне очищення потоку, відсікання шуму, віконну агрегацію, нормалізацію та відбір релевантних подій, після чого у хмару передаються вже стиснені або семантично збагачені результати. Такий підхід дозволяє одночасно зменшити затримку, знизити навантаження на хмару та раціональніше використовувати мережеві ресурси [4], [9-10], [12].

1.2.4 Обмеження edge-пристроїв

Попри значні переваги, edge computing не усуває всіх проблем обробки даних, а лише переносить частину з них на периферійний рівень. Насамперед це стосується обмеженості ресурсів edge-вузлів. На відміну від хмарних дата-центрів,

периферійні пристрої часто мають менший обсяг пам’яті, нижчу обчислювальну потужність, обмеження за енергоспоживанням та менш стабільні умови експлуатації. Це ускладнює розгортання ресурсоємних алгоритмів, великих моделей аналізу або тривалого локального зберігання даних [4-5], [9]. Особливо гостро ця проблема проявляється в IoT-середовищах, де периферійні вузли повинні забезпечувати швидкодію за умов різнорідного обладнання та непостійної якості зв’язку.

Іншим суттєвим обмеженням є гетерогенність edge-інфраструктури. У межах однієї системи можуть одночасно функціонувати шлюзи, мікросервери, промислові контролери, мережеві вузли, базові станції, одноплатні комп’ютери та спеціалізовані вбудовані рішення. Таке різноманіття ускладнює стандартизацію, оркестрацію сервісів і забезпечення сумісності програмних компонентів. У fog-парадигмі окремо підкреслюються потреби в інтероперабельності, федерації та багаторівневому керуванні [7]. У MEC-середовищі ці проблеми частково вирішуються стандартизацією API та архітектурних інтерфейсів, однак навіть там залишаються складнощі міжплатформної інтеграції та багатовендорного розгортання [6].

Окрему групу проблем становлять питання безпеки, надійності й адміністрування. На периферії мережі вузли часто фізично доступні, працюють у менш контрольованому середовищі, можуть часто змінювати стан підключення та взаємодіяти з великою кількістю зовнішніх пристроїв. У зв’язку з цим підвищуються вимоги до захисту каналів зв’язку, автентифікації пристроїв, оновлення програмного забезпечення, журналювання подій та локального резервування даних, [10], [12]. Крім того, зі збільшенням кількості edge-вузлів ускладнюється централізоване керування політиками обробки, розподіл навантаження та контроль узгодженості результатів між локальними й хмарними рівнями, [10].

Ще одним обмеженням є необхідність пошуку компромісу між локальною швидкодією та повнотою даних. Надмірне скорочення або агрегація інформації на

периферії може призвести до втрати важливих деталей, які знадобляться на наступних етапах аналітики. Натомість передавання занадто великого обсягу сирих даних нівелює переваги edge-підходу. У працях з edge computing неодноразово підкреслюється, що одним із ключових відкритих питань є визначення оптимального рівня локальної обробки: які дані потрібно фільтрувати, які агрегувати, а які передавати у первинному вигляді [4], [9-10]. Для теми цієї роботи це означає, що алгоритм попередньої обробки повинен бути не лише ефективним з погляду зниження затримок, а й достатньо обережним, щоб не погіршувати інформативність потоку даних [4], [9].

Отже, парадигма edge computing є закономірним розвитком розподілених обчислень у середовищах IoT, де критично важливими є низька затримка, контекстна обізнаність, зменшення трафіку та підвищення автономності систем. Водночас edge-підхід не усуває потреби у хмарних сервісах, а формує з ними взаємодоповнювану багаторівневу архітектуру. Саме тому подальший аналіз потокової обробки, методів агрегації та механізмів попередньої фільтрації даних має проводитися з урахуванням того, що периферійний рівень повинен забезпечувати баланс між швидкодією, точністю, стійкістю та обмеженими ресурсами [3], [7-10], [12].

Потокова обробка даних в IoT

1.3.1 Особливості потокових даних

Потокові дані є однією з базових форм подання інформації в сучасних IoT-системах, оскільки значна частина відомостей надходить від сенсорів, виконавчих пристроїв, мережевих вузлів і цифрових сервісів у безперервному або квазібезперервному режимі. На відміну від пакетної обробки, де працюють із заздалегідь сформованими наборами даних, потокова обробка орієнтована на неперервну послідовність подій, яка потенційно не має завершення та повинна аналізуватися у процесі надходження. У сучасних дослідженнях ця відмінність розглядається як принципова, оскільки для потокових систем важливими є не лише

правильність результату, а й час його отримання, порядок появи подій і механізми роботи з незавершеним потоком даних [13], [17].

Для потокових даних характерні висока швидкість надходження, змінна інтенсивність, неоднорідність структури та чутливість до часу. У середовищах IoT це проявляється особливо чітко, оскільки різні пристрої можуть генерувати часові ряди, телеметрію, подієві повідомлення, службові сигнали та метадані з різною частотою та різними форматами кодування. Крім того, такі потоки часто є розподіленими, тобто надходять не з одного джерела, а з великої кількості географічно віддалених вузлів, що додатково ускладнює забезпечення їх узгодженої обробки. Саме тому в IoT-системах потокові дані слід розглядати не просто як послідовність записів, а як динамічний, неоднорідний та часово залежний інформаційний ресурс [13].

Важливою особливістю потокових даних є те, що для них час виступає не лише службовим параметром, а повноцінною складовою семантики обробки. У потокових системах зазвичай розрізняють час виникнення події та час її обробки системою. Такий поділ є критично важливим, оскільки в реальних розподілених середовищах події можуть надходити із запізненням або поза правильним часовим порядком. Для забезпечення коректної обробки таких ситуацій сучасні моделі stream processing використовують поняття event time, processing time, а також механізми віконної обробки, тригерів і watermark-ів, які допомагають оцінювати ступінь часової повноти потоку [13].

Ще однією характерною рисою потокових даних є залежність результату від накопиченого стану системи. На практиці багато операцій потокової обробки не можуть бути реалізовані як незалежне опрацювання окремих записів, оскільки потребують агрегування, порівняння з попередніми значеннями, обчислення ковзних характеристик або виявлення шаблонів у часових інтервалах. Відповідно, сучасні stream processing-системи дедалі більше орієнтуються на stateful processing, тобто на обробку зі збереженням стану, де ключову роль відіграють вікна, локальні стани операторів і механізми їх відновлення після збоїв. Для IoT-сценаріїв це має

особливе значення, адже саме такі механізми забезпечують можливість локального виявлення подій, попередньої агрегації та підготовки даних до подальшої передачі на периферійні або хмарні вузли [14-15].

Отже, потокові дані в IoT-середовищах характеризуються безперервністю, часовою залежністю, гетерогенністю, потенційною необмеженістю та потребою у станозалежній обробці. Саме ці властивості формують вимоги до архітектур stream processing і пояснюють, чому традиційні пакетні підходи не здатні повною мірою забезпечити ефективне функціонування сучасних IoT-систем [13], [17].

1.3.2 Архітектури та моделі stream processing

У загальному випадку архітектура потокової обробки даних описується як орієнтований граф, у якому джерела даних, оператори обробки та приймачі результатів утворюють єдиний конвеєр. Подібну логіку використовували як ранні системи потокової обробки, так і сучасні distributed stream processing frameworks. У роботі про MillWheel підкреслюється, що користувач формує directed computation graph, а система забезпечує неперервний рух записів, збереження стану й відмовостійкість. Аналогічно у Flink stream- і batch-завдання розглядаються як конвеєрні fault-tolerant dataflows, що виконуються в межах єдиної моделі [14-15].

Сучасні моделі stream processing умовно можна поділити на два великі класи. Перший клас становлять системи, що обробляють події у режимі record-at-a-time, тобто поелементно, з мінімальною затримкою та явною орієнтацією на істинну потокову обробку. До цього підходу належать, зокрема, MillWheel та Apache Flink. Другий клас формують системи мікропакетної обробки, у яких потік подій ділиться на дуже короткі часові інтервали, а далі опрацьовується як послідовність малих пакетів. Класичним прикладом такого підходу є Spark Streaming, тоді як Structured Streaming у Spark розвиває декларативну модель безперервної обробки, формалізуючи потік як інкрементально оновлювану таблицю, але зберігаючи важливу роль мікропакетного виконання в більшості практичних сценаріїв [15].

Окреме місце в архітектурі stream processing займає модель часу та роботи з невпорядкованими подіями. Традиційне припущення про те, що події надходять у тому самому порядку, в якому виникають, у реальних IoT-середовищах часто не виконується через нестабільність мережі, буферизацію, повторну передачу та географічну розподіленість джерел. Саме тому сучасні архітектури потокової обробки включають механізми вікон, тригерів, watermark-ів, retractions і state management, які дозволяють працювати з out-of-order streams без втрати коректності обчислень [13].

Для розподілених IoT-систем принципово важливою є також багаторівнева архітектура потокової обробки, у межах якої частина операцій виконується на периферії, а інша частина — у хмарі або в централізованому кластері. У такій моделі джерела даних формують первинний потік, edge-вузли або шлюзи виконують попередню фільтрацію, агрегацію чи нормалізацію, а хмарні компоненти здійснюють довготривале зберігання, глобальну аналітику та міжпотокову кореляцію. Подібний підхід описується як один із перспективних напрямів розвитку distributed data stream processing, особливо в умовах поєднання edge- і cloud-ресурсів [9], [12], [16].

Таким чином, архітектури stream processing еволюціонували від локальних систем обробки подій до складних розподілених платформ, що підтримують stateful computations, часову семантику, масштабування та багаторівневе розміщення обчислень. Для задач IoT це особливо важливо, оскільки саме правильний вибір архітектурної моделі визначає, чи зможе система забезпечити достатню швидкодію, коректність результатів і стійкість до коливань навантаження, [16-17].

1.3.3 Обробка даних у реальному часі

Обробка даних у реальному часі в IoT-системах означає здатність системи аналізувати потоки подій із затримкою, достатньо малою для своєчасного виявлення станів, аномалій або керувальних ситуацій. У контексті прикладних

інформаційних систем це зазвичай не означає жорстких часових гарантій на рівні апаратного реального часу, а передбачає мінімізацію end-to-end latency між виникненням події, її передаванням, обробкою та формуванням реакції системи. Саме тому в сучасній літературі з потокової аналітики поняття реального часу тісно пов’язується з показниками latency, throughput, timeliness та consistency результатів,, [18].

Ключовою умовою коректної обробки в реальному часі є правильне трактування часової семантики подій. Якщо система орієнтується лише на момент обробки, то за наявності мережевих затримок або повторної доставки результати можуть бути неповними або семантично спотвореними. Саме тому модель Dataflow та подальші дослідження watermark-ів підкреслюють, що для потокових систем необхідно відокремлювати event time від processing time, а також використовувати windowing та trigger semantics, які дозволяють досягти прийнятного компромісу між повнотою результату, затримкою та обчислювальними витратами [13].

Не менш важливим елементом реального часу є гарантія узгодженості результатів за умов відмов, повторної доставки та масштабування. Для цього stream processing-системи реалізують різні режими гарантій, зокрема at-most-once, at-least-once та exactly-once. У практичних архітектурах для досягнення сильніших гарантій використовуються механізми checkpointing, журнали подій, відновлення стану та транзакційні схеми запису результатів. Водночас підвищення надійності зазвичай супроводжується збільшенням накладних витрат, а отже, питання реального часу завжди розглядається разом із питанням компромісу між швидкодією та коректністю [14], [18].

Для IoT-систем обробка в реальному часі має особливу цінність через те, що сенсорні дані часто відображають фізичні процеси, стан обладнання або середовища, який змінюється неперервно. Відповідно, запізнілий результат може бути не просто менш корисним, а фактично непридатним для прийняття рішень. Саме тому у промисловому IoT, транспортних системах, моніторингу

інфраструктури та системах підтримки безпеки дедалі ширше застосовують розподілені конвеєри обробки в реальному часі, де частина реакції формується ще на периферійному рівні, без обов'язкового очікування відповіді від хмарної платформи [4], [9], [12].

Отже, обробка даних у реальному часі в IoT — це не лише питання швидкості виконання операторів, а комплексна властивість системи, що охоплює часову семантику подій, механізми підтримки стану, правила формування результату, гарантії обробки та архітектурне розміщення обчислень. Саме така багатокомпонентність робить stream processing ключовим інструментом для сучасних IoT- і edge-рішень [13], [18].

1.3.4 Проблеми масштабованості, затримок і надійності

Попри значні переваги потокової обробки, її практична реалізація в IoT-середовищах пов'язана з низкою складних технічних проблем. Насамперед це стосується масштабованості. У реальних системах навантаження на потік може змінюватися нерівномірно, а кількість подій — різко зростати унаслідок сплесків активності, збільшення числа пристроїв або змін у частоті вимірювання. За таких умов система має підтримувати горизонтальне масштабування, перерозподіл навантаження між вузлами, коректну міграцію стану операторів і збереження часової семантики під час зміни ступеня паралелізму. Саме ці аспекти в сучасній літературі визначаються як центральні проблеми еластичності distributed stream processing systems, [16].

Другою критичною проблемою є затримка обробки. Навіть якщо система здатна підтримувати високий throughput, це не гарантує малого часу відгуку. Затримка формується на кількох рівнях одночасно: під час доставки подій мережею, буферизації у брокерах, очікування в чергах операторів, серіалізації даних, запису та відновлення стану, а також унаслідок контрольних точок і механізмів backpressure. Експериментальні дослідження показують, що під інтенсивним навантаженням механізми керування потоком і пам'яттю самі можуть

ставати джерелом зростання latency, особливо для stateful pipelines та data-intensive workloads, [18].

Третьою фундаментальною проблемою є надійність. Оскільки потоки даних є потенційно нескінченними, а обробка виконується у розподіленому середовищі, система повинна зберігати коректність результатів навіть у разі відмов окремих вузлів, перевантажень, затримок мережі або повторної доставки повідомлень. Для цього застосовують checkpointing, журналювання, реплікацію, replay-based recovery, збереження локального стану та транзакційні механізми взаємодії з зовнішніми системами. Проте практична реалізація цих підходів потребує ретельного балансу між відмовостійкістю та продуктивністю, оскільки посилення гарантій обробки майже завжди збільшує накладні витрати [14-15], [18].

Для IoT-середовищ перелічені проблеми посилюються обмеженнями мережі та периферійних вузлів. Потоки можуть надходити з нестабільних бездротових мереж, окремі сегменти інфраструктури можуть тимчасово втрачати зв’язок, а edge-пристрої часто мають менші обчислювальні та пам’яттєві ресурси, ніж серверні вузли. У таких умовах особливо важливими стають механізми локальної буферизації, попередньої агрегації, адаптивного зменшення навантаження, вибіркового передавання даних і раннього виявлення аномалій. Українські дослідження також підкреслюють, що потоковий аналіз у реальному часі є не лише інструментом аналітики, а й засобом підвищення відмовостійкості розподіленої інфраструктури завдяки ранньому виявленню проблем і можливості швидкої реакції [12], [16].

Отже, масштабованість, затримки та надійність слід розглядати як взаємопов’язані характеристики потокової обробки, а не як ізольовані вимоги. Для систем IoT це означає, що ефективний підхід до stream processing має не лише забезпечувати високу пропускну здатність, а й підтримувати коректну часову семантику, стійкість до збоїв і раціональне використання периферійних ресурсів. Саме ця сукупність вимог і створює підґрунтя для подальшого розгляду методів агрегації та попередньої обробки даних на edge-рівні [9], [12], [16], [18].

1.4 Методи агрегації та попередньої обробки даних

1.4.1 Сутність агрегації даних

Агрегація даних у системах IoT розглядається як процес узагальнення, поєднання або зведення множини первинних вимірювань до компактнішого й змістовно придатного представлення. У класичних роботах із сенсорних мереж агрегація трактується як in-network processing, тобто виконання частини операцій безпосередньо всередині мережі, а не лише на централізованому сервері. Такий підхід дозволяє скорочувати кількість переданих повідомлень, зменшувати енергоспоживання та знижувати навантаження на канали зв’язку, що особливо важливо для ресурсно обмежених IoT-середовищ [19-20].

У сучасних IoT-архітектурах агрегація виконує не лише функцію стискання інформації, а й виступає засобом підготовки даних до подальшого аналізу. Вона дає змогу перетворити великий потік сирих сенсорних значень на компактні статистичні характеристики, події або індикатори стану системи. У цьому розумінні агрегація є проміжною ланкою між первинним збором даних і складнішою аналітикою, оскільки саме вона формує базу для виявлення трендів, аномалій, перевищення порогів або змін у поведінці об’єкта спостереження [20].

Для теми даної роботи сутність агрегації полягає насамперед у раціональному зменшенні інформаційного потоку без критичної втрати змісту. Це означає, що агрегація на edge-рівні має виконувати дві взаємопов’язані функції: по-перше, скорочувати обсяг передаваних у хмару даних, а по-друге, зберігати ті властивості потоку, які є важливими для подальшого прийняття рішень. Саме тому в IoT-системах агрегація не може розглядатися як суто механічне усереднення значень; вона має враховувати природу сенсорних вимірювань, часовий контекст і цілі конкретного застосування [19-21].

1.4.2 Основні методи агрегації даних

Найпростішими та найбільш уживаними методами агрегації є обчислення базових статистичних характеристик: суми, кількості записів, середнього значення, мінімуму та максимуму. Такі операції є природними для сенсорних потоків, оскільки дозволяють швидко узагальнювати значення за певний інтервал часу або для певної групи вузлів. У класичних системах типу TAG саме подібні агрегати розглядалися як базові building blocks для розподіленої обробки в сенсорних мережах [19]. У сучасних оглядових роботах також підкреслюється, що прості агрегати залишаються основою більшості практичних рішень через низьку обчислювальну складність та придатність до виконання на ресурсно обмежених вузлах [20].

Окрім елементарних статистик, у практиці IoT широко застосовуються методи агрегування, орієнтовані на представлення розподілу даних. До них належать гістограми, частотні таблиці, оцінки квантилів та інші synopsis-подання, що дозволяють компактно описувати структуру потоку. На відміну від простого середнього, такі методи краще відображають неоднорідність потоку, варіативність значень і локальні концентрації даних. У дослідженнях з approximate query processing показано, що гістограми, wavelet-синопсиси, вибірки та sketches можуть бути ефективною основою для наближених відповідей на агрегатні запити в середовищах з великими обсягами або високою швидкістю надходження даних.

Для IoT-середовищ доцільно розрізняти точну та наближену агрегацію. Точна агрегація передбачає використання всіх доступних значень у вікні або групі, тоді як наближена базується на скороченому поданні потоку, наприклад вибірках, гістограмах або компактних статистичних структурах. Перевагою точного підходу є максимальна інформативність, однак він потребує більше пам’яті, пропускної здатності й часу обробки. Наближена агрегація, навпаки, дозволяє істотно зменшити витрати ресурсів, але вводить похибку, величина якої має контролюватися відповідно до вимог конкретного застосування [20].

З огляду на тему дипломної роботи, найбільший практичний інтерес становлять саме ті методи агрегації, які поєднують простоту реалізації з можливістю суттєвого зниження обсягу даних. До таких методів належать усереднення за часовими інтервалами, обчислення мінімумів і максимумів, підрахунок кількості подій, формування гістограм частот та порогове виділення суттєвих змін. Саме вони можуть бути реалізовані на edge-пристроях без надмірних витрат пам’яті та процесорного часу [19-20].

1.4.3 Методи фільтрації, очищення та нормалізації даних

Попередня обробка сенсорних потоків неможлива без фільтрації та очищення даних, оскільки реальні IoT-потоки часто містять шуми, пропущені значення, дублікати, аномальні вимірювання та часову неузгодженість. Огляди з data quality в IoT підкреслюють, що якість даних безпосередньо впливає на достовірність аналітики й коректність рішень, прийнятих на основі цих даних. Серед типових причин погіршення якості називають збої сенсорів, нестабільність мережі, вплив зовнішнього середовища та неоднорідність форматів даних.

Фільтрація даних у IoT-системах зазвичай спрямована на зменшення впливу шуму та випадкових коливань. У найпростішому випадку застосовують згладжувальні або порогові фільтри, а також низькочастотні чи медіанні перетворення, які зменшують чутливість до короткочасних викидів. В українських навчальних матеріалах із обробки інформації в автоматизованих системах підкреслюється, що попередня обробка сигналів включає фільтрацію, корекцію шумів і нормалізацію, тобто фактично виконує роль підготовчого шару перед аналітичним рівнем [23]. Для IoT це означає, що частина фільтрації може бути реалізована ще до передачі потоку у верхні шари системи, [23].

Очищення даних включає видалення або виправлення записів, які знижують надійність подальшого аналізу. До таких операцій належать усунення дублікатів, обробка пропусків, виявлення та маркування аномальних значень, перевірка допустимих меж вимірювання та відновлення часової послідовності подій.

Систематичні огляди з data quality для CPS та IoT вказують, що в індустріальних застосуваннях важливими є не лише самі техніки очищення, а й метрики оцінювання якості, правила data repair і механізми постійного моніторингу якості потоку. Отже, очищення доцільно розглядати як безперервний процес, а не як одноразову передобробку.

Нормалізація даних спрямована на приведення неоднорідних вимірювань до узгодженого масштабу, формату або представлення. У середовищах IoT це особливо важливо, оскільки дані можуть надходити від різнорідних датчиків, використовувати різні одиниці вимірювання, діапазони значень та формати кодування. Тому нормалізація може охоплювати не лише числове масштабування, а й уніфікацію схем даних, перетворення одиниць, синхронізацію часових міток і приведення повідомлень до єдиного внутрішнього формату. У сучасних оглядах з IoT-аналітики й edge processing підкреслюється, що такі операції доречно виконувати вже на проміжному або периферійному рівні, оскільки це полегшує подальший аналіз і зменшує потребу в централізованій обробці сирих даних, [23].

1.4.4 Методи зменшення обсягу даних

У задачах IoT та edge computing зменшення обсягу даних є однією з центральних цілей попередньої обробки. Потреба в таких методах зумовлена обмеженою пропускною здатністю каналів зв’язку, енергетичними обмеженнями кінцевих вузлів і небажаністю передавати до хмари весь сирий потік. У літературі з data reduction для IoT показано, що зменшення даних може виконуватися як на рівні сенсорів, так і на рівні шлюзів або периферійних вузлів, причому ефективні рішення часто поєднують кілька шарів редукції — фільтрацію, агрегацію та злиття даних [21].

Одним із найпоширеніших підходів є стиснення даних. Для часових рядів, що домінують у IoT-середовищах, використовують як безвтратні, так і втратні методи стиснення. Безвтратне стиснення доцільне тоді, коли критично важливо зберегти повну точність вимірювань, тоді як втратне стиснення є виправданим у

випадках, коли допустимою є керована похибка заради істотного скорочення обсягу передавання або зберігання. У сучасних оглядах зі стиснення часових рядів підкреслюється, що ефективність конкретного методу залежить від властивостей сигналу, допустимого рівня похибки та обчислювальних ресурсів пристрою.

Іншим важливим методом зменшення даних є sampling, тобто відбір частини записів потоку замість повної обробки всіх елементів. Для потоків невідомої або необмеженої довжини класичним рішенням є reservoir sampling, який дозволяє підтримувати репрезентативну вибірку фіксованого розміру в один прохід через потік. Такий підхід особливо корисний у випадках, коли повне зберігання або передавання даних є недоцільним, але потрібно зберегти можливість приблизної оцінки розподілу, середніх характеристик або поведінки потоку в цілому [22].

Окрім стиснення та вибірки, у потокових системах активно використовують synopsis-методи: гістограми, wavelet-подання, sketches та інші компактні структури, що зберігають важливі статистичні властивості даних. Їхня цінність полягає в тому, що вони не лише скорочують обсяг даних, а й дозволяють напряму виконувати наближені аналітичні запити без звернення до повного набору вимірювань. У контексті IoT це означає можливість передавати до хмари не потік у первинному вигляді, а вже узагальнений опис його основних характеристик. Відповідно, для периферійних алгоритмів особливо перспективними є комбіновані схеми, у яких фільтрація, локальне злиття даних, агрегація та synopsis-подання працюють спільно [21].

1.4.5 Віконні методи обробки потоків даних

Оскільки потік даних у загальному випадку є необмеженим, агрегування по всій його довжині практично неможливе. Саме тому в stream processing застосовують віконні методи, які обмежують обчислення певним часовим або кількісним фрагментом потоку. У моделі Dataflow та споріднених системах вікно розглядається як базова абстракція для перетворення нескінченного потоку на

скінченні підмножини, над якими вже можна коректно виконувати агрегатні операції [13].

Найпоширенішими є tumbling windows та sliding windows. Tumbling window — це неперекривне вікно фіксованої довжини, яке ділить часову шкалу на послідовні інтервали без перетину. Такий варіант є зручним для періодичного підбиття підсумків, наприклад для обчислення середнього значення за кожну хвилину або годинй максимум температури. Sliding window, навпаки, має фіксовану довжину, але зсувається через менший інтервал, унаслідок чого сусідні вікна частково перекриваються. Це дозволяє отримувати плавніші оцінки та краще виявляти короткочасні зміни, однак збільшує кількість повторних обчислень [13].

У практиці також використовують count-based windows, де межі визначаються кількістю записів, а не часовим інтервалом, а також session windows, що групують події за періодами активності, розділеними паузами. Однак для задач периферійної обробки телеметрії найбільшу практичну цінність мають саме фіксовані tumbling і sliding windows, оскільки вони легко реалізуються, дозволяють контролювати затримку та природно поєднуються з алгоритмами усереднення, мін/макс-агрегації, порогового аналізу та компресії потоку [13].

Коректна реалізація віконної обробки в розподілених середовищах вимагає врахування часової семантики подій. Якщо події надходять із запізненням або поза порядком, то межі вікна не можна визначати лише за моментом фактичної обробки. Саме тому сучасні stream processing-системи використовують event time та watermark-и, які дозволяють оцінювати часову повноту потоку й приймати рішення про завершення вікна або допустимість пізніх подій [13]. Для задач IoT на edge-рівні це має особливе значення, оскільки нестабільність мережі та нерівномірність доставки повідомлень є типовими явищами [13].

Таким чином, віконні методи обробки є фундаментом практичної реалізації агрегації у потокових IoT-системах. Саме вони забезпечують обмеження пам’яті, керовану затримку та можливість виконувати статистичне узагальнення над

неперервним потоком даних. Для теми даної дипломної роботи це означає, що алгоритм агрегації та попередньої обробки на периферійних пристроях доцільно будувати саме на основі віконної логіки, оскільки вона найкраще відповідає задачам зменшення навантаження на мережу та хмарну інфраструктуру без втрати часової релевантності результатів [13].

1.5 Оптимізація передачі та обробки даних у IoT-системах і обґрунтування розробки edge-алгоритму

1.5.1 Проблема затримок і перевантаження хмарної інфраструктури в IoT-системах

Однією з ключових проблем сучасних IoT-систем є суперечність між безперервним зростанням кількості підключених пристроїв і обмеженнями централізованої хмарної моделі обробки даних. У традиційному підході значна частина телеметрії передається до віддалених дата-центрів, де виконується її накопичення, аналіз і прийняття рішень. Така модель забезпечує високу масштабованість і централізоване адміністрування, однак водночас збільшує довжину маршруту проходження даних, а отже — і затримку між моментом виникнення події та моментом її обробки. Для систем, орієнтованих на реальний час, кіберфізичні процеси або промисловий моніторинг, це стає суттєвим обмеженням [3-4], [24-25].

Проблема затримок у cloud-centric архітектурі пов’язана не лише з фізичною віддаленістю хмарного центру обробки, а й із сукупністю проміжних етапів: мережевою передачею, буферизацією, маршрутизацією, повторною доставкою повідомлень, чергами обробки та централізованим доступом до сховищ. У міру зростання числа пристроїв і частоти надходження даних ці накладні витрати накопичуються, а хмарна інфраструктура починає виступати вузьким місцем, особливо тоді, коли значна частина сирих даних не потребує довготривалого зберігання, але все одно транспортується у центр обробки [9], [16], [25].

Ще однією складовою проблеми є перевантаження транспортної та хмарної інфраструктури надлишковими вимірюваннями. Значна частина IoT-потоків містить висококорельовані, повторювані або малоінформативні значення, наприклад тривалі відрізки стабільних телеметричних даних, які майже не змінюються в часі. Передавання всього потоку без попереднього відбору або узагальнення призводить до нераціонального використання пропускної здатності мережі, сховищ та обчислювальних ресурсів хмари. Українські дослідники також підкреслюють, що для IoT та IoT-Fog-Cloud архітектур централізована модель без проміжного рівня обробки поступово втрачає ефективність через зростання навантаження та вимог до швидкодії [11-12], [33].

Отже, для IoT-систем проблема затримок і перевантаження хмарної інфраструктури має системний характер. Вона зумовлена не лише обсягом даних, а й самою архітектурною логікою централізованої обробки. Саме тому питання оптимізації передачі даних слід розглядати не як допоміжне, а як одне з базових завдань проєктування сучасних IoT-рішень [4], [9], [25].

1.5.2 Методи оптимізації передачі даних у IoT

Оптимізація передачі даних у IoT зазвичай реалізується через поєднання кількох груп методів: локальної попередньої обробки, агрегації, редукції потоку, зміни режиму передавання та ієрархічного розподілу обчислень між пристроєм, edge-вузлом і хмарою. Найбільш базовим підходом є виконання первинної обробки безпосередньо на сенсорному або периферійному рівні, коли частина даних фільтрується, агрегується або нормалізується ще до потрапляння у транспортну мережу. Це дозволяє скоротити кількість переданих повідомлень, зменшити навантаження на хмару та прискорити локальну реакцію системи [9], [12], [21].

Другою важливою групою методів є редукція обсягу даних за допомогою стиснення, вибірки або synopsis-подань. У попередньому підрозділі було показано, що для потокових сенсорних даних можуть застосовуватися безвтратні й втратні схеми стиснення, sampling, гістограми, sketches та інші компактні форми

представлення. З погляду оптимізації передачі ці підходи є ефективними тоді, коли система може дозволити собі контрольовану втрату деталізації або працює з даними, корисність яких визначається не окремим вимірюванням, а узагальненими характеристиками потоку [21].

Третій напрям оптимізації пов’язаний з адаптацією режиму передавання до динаміки самого сигналу. У цьому випадку дані не передаються з однаковою періодичністю, а надсилаються лише тоді, коли зміна значення, похибка прогнозу або виявлена подія перевищує певну межу. Такі методи дозволяють уникати надсилання великої кількості майже ідентичних повідомлень під час стабільного стану об’єкта спостереження. Вони особливо ефективні для автономних або батарейних пристроїв, де витрати на радіопередачу є одним із головних джерел енергоспоживання [26–28].

Нарешті, окремий клас методів оптимізації базується на архітектурному рознесенні функцій між шарами IoT–edge–cloud. У такій моделі потік не просто скорочується, а логічно розподіляється: частина обчислень виконується на пристрої або edge-шлюзі, а до хмари потрапляють лише агреговані показники, події, аналітичні ознаки або історичні дані для довготривалого зберігання та глобальної аналітики. Саме цей підхід нині розглядається як один із найбільш перспективних для систем із великим числом джерел даних і вимогами до низької затримки [4], [9-10], [16], [25].

1.5.3 Event-driven та threshold-based підходи до передавання даних

Періодичне передавання даних є найпростішим способом організації телеметрії, однак у багатьох IoT-сценаріях воно є неефективним, оскільки не враховує реальну динаміку сигналу. Якщо значення тривалий час залишаються стабільними, система однаково продовжує передавати вимірювання через фіксовані інтервали, створюючи зайвий трафік і додаткове енергоспоживання. Саме тому альтернативою періодичному режиму стали event-driven підходи, у яких

ініціатором передавання виступає не сам факт настання чергового часового інтервалу, а поява зміни, яка має прикладне значення для системи [26].

Event-driven передавання зазвичай реалізується через правила, що визначають, коли потік слід вважати достатньо “важливим” для надсилання. Одним із найпоширеніших варіантів є threshold-based підхід, коли передача відбувається лише тоді, коли абсолютна зміна вимірювання, накопичена помилка або відхилення від прогнозованого значення перевищують заданий поріг. У практиці IoT це дає змогу уникати передачі надлишкових, майже незмінних даних та концентрувати ресурси мережі на дійсно інформативних змінах стану [26–28].

Ефективність event-driven підходу підтверджується експериментально. У дослідженні C. Santos, J. A. Jiménez та F. Espinosa, присвяченому event-based sensing у smart city-сценарії, показано, що такі техніки можуть забезпечувати до 50% економії споживання сенсорного вузла порівняно з класичним періодичним вимірюванням [26]. У роботі S. Suryavansh та співавт. запропоновано протокол Ambrosia, який використовує конфігурований поріг похибки для скорочення передавання даних без погіршення придатності даних для прикладних застосувань; у реальному розгортанні на базі LoRa та BLE автори зафіксували 60% скорочення передавання та двократне збільшення часу роботи сенсорних вузлів [27].

Водночас статичний поріг не завжди є оптимальним, оскільки динаміка IoT-сигналів може змінюватися в часі. Саме тому сучасні дослідження переходять до adaptive threshold-моделей. У роботі H. Zhang та співавт. запропоновано автономний метод редукції IoT-даних з адаптивним порогом, який коригується залежно від concept drift; автори показують, що поріг має зменшуватися при виявленні змін у даних і збільшуватися за відсутності таких змін. За їхніми експериментами це дало в середньому 11,7% покращення точності при тому самому рівні редукції або 17,3% покращення редукції при тій самій точності [28].

Отже, event-driven і threshold-based підходи є одними з найперспективніших механізмів оптимізації передачі даних у IoT. Вони особливо добре поєднуються з

edge-обробкою, оскільки рішення про передавання може прийматися локально, без залучення хмари, що додатково знижує затримку та навантаження на транспортну інфраструктуру [26–28].

1.5.4 Порівняння централізованої та периферійної моделей обробки даних

Централізована cloud-модель і периферійна edge-модель не є взаємовиключними, однак їхні сильні сторони суттєво відрізняються. Хмарні обчислення забезпечують централізоване зберігання, високу еластичність, масштабованість та зручність інтеграції з аналітичними сервісами великого масштабу. У свою чергу, edge-підхід переносить обчислення ближче до джерела даних, що дає змогу скоротити затримку, зменшити обсяг переданого трафіку та підвищити автономність локального сегмента системи [3-4], [7], [9], [25].

Для IoT-середовищ головною перевагою edge-підходу є можливість виконувати попередню обробку, агрегацію, виявлення подій і локальну реакцію без необхідності передавати весь потік у віддалену хмару. Це особливо важливо для промислових систем, інфраструктурного моніторингу, транспортних сервісів і застосувань реального часу. Водночас cloud-модель зберігає свою цінність для глобального аналізу, довготривалого архівування, навчання моделей та міжоб’єктної кореляції даних. Саме тому сучасні дослідження дедалі частіше розглядають не жорстке протиставлення cloud та edge, а їхню кооперацію в межах гібридної або континуальної архітектури [9-10], [16], [25].

Українські публікації, присвячені fog/cloud та IoT-Fog-Cloud архітектурам, також підтверджують, що винесення частини обчислень на проміжний рівень дозволяє зменшити час затримки, підвищити ефективність локальної обробки та краще пристосувати систему до вимог IoT-додатків. Водночас централізована модель залишається важливою там, де потрібні великі централізовані ресурси й наскрізне адміністрування [12], [33].

Таким чином, порівняння cloud і edge свідчить, що для задачі зменшення затримок і навантаження на хмарну інфраструктуру найбільш доцільним є не повне

відмовлення від хмари, а перенесення частини функцій попередньої обробки на периферійний рівень. Саме це дозволяє поєднати сильні сторони обох підходів: локальну швидкодію та глобальну масштабованість [4], [9-10], [25].

1.5.5 Огляд сучасних платформ і рішень edge computing

Сучасний ринок edge computing включає як комерційні, так і відкриті платформи, що підтримують розгортання локальної аналітики, маршрутизацію повідомлень, керування пристроями та інтеграцію з хмарними сервісами. Серед комерційних рішень важливе місце посідає AWS IoT Greengrass, який, за офіційною документацією AWS, надає локальні обчислення, обмін повідомленнями, керування даними, синхронізацію та можливості ML inference на edge-пристроях; система дозволяє збирати та аналізувати дані ближче до місця їх генерування й автономно реагувати на локальні події [29]. У поточній екосистемі Microsoft доступніAzure IoT Edge 1.5 LTS як device-focused runtime для контейнеризованих застосунків на пристрої та Azure IoT Operationsяк unified data plane for the edge на Azure Arc-enabled Kubernetes, що орієнтований на промислові сценарії, edge-native MQTT, data flows і локальну нормалізацію даних [30].

Серед відкритих платформ варто виділити KubeEdge, EdgeX Foundry та Open Horizon. KubeEdge позиціонується як open source система, що розширює нативні можливості Kubernetes до edge-вузлів і забезпечує базову інфраструктуру для мережевої взаємодії, розгортання застосунків та синхронізації метаданих між cloud і edge [31]. EdgeX Foundry, своєю чергою, є вендоронейтральним відкритим фреймворком для IoT edge computing з інтероперабельною, hardware- та OS-agnostic платформою, орієнтованою на побудову екосистеми plug-and-play компонентів [32]. Open Horizon зосереджується на іншому аспекті — автономному керуванні життєвим циклом контейнеризованих навантажень і пов’язаних ML-активів на великих розподілених парках edge-вузлів.

Для задач потокової аналітики особливо цікаві рішення, що безпосередньо підтримують stream processing на периферії. Прикладом є LF Edge eKuiper, який

офіційно позиціонується як lightweight IoT data analytics and stream processing engine для resource-constrained edge devices. Документація eKuiper підкреслює, що платформа надає SQL- або graph-based правила, підтримує ETL, агрегацію, joins і кілька типів вікон при невеликому обсязі пакета й пам’яті, що робить її особливо близькою до задач локальної аналітики IoT-потоків. Це важливо саме для цієї дипломної роботи, оскільки демонструє практичну затребуваність легких механізмів обробки потоку безпосередньо на edge-рівні.

Узагальнюючи, можна стверджувати, що сучасні edge-платформи досить добре вирішують інфраструктурні завдання: розгортання компонентів, керування пристроями, інтеграцію з хмарою, локальне виконання контейнерів, маршрутизацію повідомлень і підтримку потокових правил. Однак вони здебільшого надають саме платформні механізми, тоді як конкретна логіка агрегації, фільтрації, редукції потоку чи прийняття рішення про передавання часто залишається завданням розробника прикладного рішення [29]. Це є важливим аргументом на користь розробки власного алгоритмічного підходу поверх наявної edge-інфраструктури.

1.5.6 Баланс між точністю, ефективністю та обґрунтування необхідності розробки власного алгоритму

Будь-яка оптимізація передачі даних у IoT-системі фактично є задачею пошуку компромісу між кількома взаємопов’язаними цілями: зменшенням затримки, скороченням обсягу переданих даних, зниженням навантаження на мережу й хмару, енергетичною ефективністю, а також збереженням достатньої інформативності потоку. Якщо система занадто агресивно скорочує потік, то може втратити важливі локальні зміни, короткочасні аномалії або дрібні відхилення, які мають значення для подальшого аналізу. Якщо ж вона передає надто багато сирих даних, то нівелює переваги edge-підходу. Саме ця суперечність проходить через більшість сучасних досліджень у сфері stream processing, data reduction та edge analytics [13], [21], [25].

Threshold-based і prediction-based підходи добре демонструють цю проблему. З одного боку, поріг дозволяє гнучко контролювати кількість передач, а prediction-based suppression дає змогу не передавати значення, якщо вони достатньо добре відновлюються на приймальному боці. З іншого боку, вибір надто великого порогу підвищує редукцію ціною втрати точності, а надто малого — робить редукцію майже непомітною. Саме тому Suryavansh та співавт. використовують конфігурований error threshold [27], а Zhang та співавт. переходять до adaptive threshold-механізму, який автоматично підлаштовується під concept drift і зміну динаміки даних [28].

Аналіз платформних рішень також дозволяє зробити важливий висновок. Існуючі edge-платформи — AWS IoT Greengrass, Azure IoT Edge, Azure IoT Operations, KubeEdge, EdgeX Foundry, Open Horizon, eKuiper — надають потужне середовище для розгортання сервісів, потокових правил, маршрутизації повідомлень і взаємодії з хмарою, однак не пропонують єдиного універсального алгоритму, який одночасно оптимально вирішує задачу агрегації, очищення, відбору подій і зниження мережевого навантаження для ресурсно обмежених edge-пристроїв. Це твердження є узагальненням на основі аналізу документації та літератури: платформи забезпечують інфраструктуру та інструменти, а вибір конкретної логіки попередньої обробки залишається на рівні прикладного проєктування [29].

З урахуванням теми дипломної роботи це означає, що доцільним є розроблення власного edge-алгоритму, який поєднуватиме кілька механізмів одночасно: локальну агрегацію, попередню фільтрацію та очищення, віконну обробку потоків, а також адаптивне рішення щодо передачі даних у хмару. Такий алгоритм має бути достатньо легким для виконання на периферійному пристрої, але водночас досить інформативним, щоб не погіршувати якість подальшої аналітики. Саме в цьому полягає логічний перехід від теоретичного аналізу до постановки задачі розроблення власного методу, який буде розглянуто в наступному розділі [9], [12-13], [21], [27-28].

РОЗДІЛ 2

ПОСТАНОВКА ЗАДАЧІ ТА МОДЕЛЮВАННЯ IoT-СЕРЕДОВИЩА

2.1 Формалізація задачі дослідження

2.1.1 Вхідні дані та обмеження системи

Вхідними даними розроблюваної системи є часові ряди вимірювань, що надходять від гетерогенного набору IoT-датчиків, розгорнутих у межах контрольованого середовища. У рамках дослідження розглядаються три типи датчиків: температурні, датчики вологості та вібраційні. Кожен тип характеризується власною частотою дискретизації, статистичними властивостями значень та розміром мережевого пакету.

Температурні датчики формують потоки з частотою 1 Гц, значення яких відповідають нормальному розподілу N(22.35; 6.45). Датчики вологості генерують дані з частотою 0.5 Гц, розподіл значень описується бета-розподілом Beta(37.2; 55.0), масштабованим до діапазону [0; 100] відсотків. Вібраційні датчики мають найвищу частоту дискретизації — 10 Гц — і формують потоки, що підпорядковуються логнормальному розподілу LogNormal(0; 0.5), характерному для процесів з важкими хвостами. Параметри розподілів калібровано на основі реальних вимірювань Intel Lab Dataset. Зведена характеристика вхідних потоків наведена у таблиці 2.1.

Таблиця 2.1 — Характеристики вхідних потоків даних IoT-датчиків

Тип датчика	**Фізична величина**	**Частота дискретизації, Гц**	**Розподіл значень**	**Розмір пакету, байт**
Температурний	Температура, °C	1	N(22.35; 6.45)	64
Датчик вологості	Відносна вологість, %	0.5	Beta(37.2; 55.0)×100	64
Вібраційний	Прискорення, g	10	LogNormal(0; 0.5)	128

Ресурсна модель периферійного вузла визначається сукупністю апаратних обмежень цільового пристрою. Обчислювальний профіль включає чотириядерний процесор з тактовою частотою 1.2 ГГц та оперативну пам'ять обсягом 512 МБ. Для обробки вхідних пакетів використовується черга обмеженої місткості: максимальний розмір черги складає 1000 пакетів, максимально допустимий час відгуку — 100 мс. Перевищення цих значень призводить до відкидання пакетів. Ресурсні обмеження периферійного вузла наведено у таблиці 2.2.

Таблиця 2.2 — Ресурсні обмеження периферійного вузла

Параметр	**Позначення**	**Значення**
Кількість ядер CPU	N_cpu	4
Тактова частота	f_cpu	1.2 ГГц
Обсяг оперативної пам'яті	M_ram	512 МБ
Максимальний розмір черги	Q_max	1000 пакетів

Максимальний час відгуку	T_max	100 мс

Продовження табл. 2.2

2.1.2 Критерії ефективності алгоритму

Для оцінювання якості розроблюваного алгоритму агрегації визначено чотири групи метрик, що відображають різні аспекти функціонування системи: часові характеристики, мережеве навантаження, точність збереження корисного сигналу та ресурсоємність обробки.

Першою ключовою метрикою є наскрізна затримка L_e2e (end-to-end latency) — проміжок часу від моменту генерації вимірювання датчиком до моменту його отримання хмарною платформою. Ця метрика є інтегральною і включає затримку локальної передачі від датчика до edge-вузла, час обробки на вузлі та затримку мережевої передачі до хмари. Цільовий показник: $L_{e2e} \leq 200$ мс для температурних і вологісних датчиків та $L_{e2e} \leq 50$ мс для вібраційних датчиків.

Другою метрикою є обсяг трафіку V — кількість байт, переданих від периферійного вузла до хмари за одиницю часу. Ефективний алгоритм агрегації має суттєво зменшити цей показник порівняно зі сценарієм прямої передачі сирих даних без обробки на вузлі.

Третьою метрикою є похибка агрегації ε, що характеризує ступінь відхилення агрегованих даних від вихідного потоку вимірювань. Похибка визначається як середньоквадратична різниця між сирими значеннями та відповідними виходами алгоритму агрегації. Допустиме значення похибки ε_{max} встановлюється як обмеження задачі оптимізації і задається окремо для кожного типу датчиків залежно від вимог застосування.

Четвертою групою метрик є показники ресурсоємності: завантаженість процесора U_cpu (у відсотках від максимальної) та обсяг використаної оперативної пам'яті M_used. Алгоритм вважається прийнятним, якщо $U_{cpu} \leq 80\%$ та $M_{used} \leq 0.8 \cdot M_{ram}$. Зведений перелік метрик ефективності наведено у таблиці 2.3.

Таблиця 2.3 — Метрики ефективності алгоритму агрегації

Метрика	**Позначення**	**Одиниці**	**Цільовий показник**
Наскрізна затримка	L_e2e	мс	≤ 200 (темп./вол.), ≤ 50 (вібр.)
Обсяг трафіку	V	байт/с	мінімум
Похибка агрегації	ε	—	≤ ε_max
Завантаженість CPU	U_cpu	%	≤ 80
Використання RAM	M_used	МБ	≤ 409.6

2.1.3 Математична постановка задачі

Нехай $x = \{x_1, x_2, \ldots, x_n\}$ — дискретний часовий ряд вимірювань одного датчика, де $x_t \in \mathbb{R}$ — значення у момент часу t, а N — загальна кількість спостережень. Алгоритм агрегації A перетворює вхідний потік x на стиснену послідовність $\hat{x} = A(x, W, \theta)$, де $W \in \mathbb{N}$ — розмір ковзного вікна обробки, $\theta \in \mathbb{R}^+$ — порогове значення передачі даних.

Наскрізна затримка L_e2e є сумою трьох складових:

$$L_{e2e} = L_{local} + L_{proc} + L_{net} \qquad (2.1)$$

де L_local — затримка локального каналу (датчик → edge-вузол), L_proc — час обробки пакету на edge-вузлі, L_net — затримка мережевого каналу (edge-вузол → хмара).

Похибка агрегації визначається як середньоквадратична різниця між сирими та агрегованими значеннями:

$$\varepsilon(W, \theta) = (1/N) \cdot \Sigma_{t=1}^{N} (x_t - \hat{x}_t)^2 \quad (2.2)$$

де $\hat{x}_t$ — відповідне агреговане значення для моменту t.

Обсяг трафіку V(W, θ) визначається кількістю пакетів, переданих до хмари, помноженою на середній розмір пакету s:

$$V(W, \theta) = (|\hat{x}| / N) \cdot s \quad (2.3)$$

де $|\hat{x}|$ — кількість переданих агрегованих значень.

Задача оптимізації параметрів алгоритму формулюється як мінімізація зваженої суми наскрізної затримки та обсягу трафіку за умови обмеження на допустиму похибку агрегації та ресурси вузла:

$$\min F(W, \theta) = \alpha \cdot L\_e2e(W, \theta) + \beta \cdot V(W, \theta) \quad (2.4)$$

за умов:

$$\varepsilon(W, \theta) \leq \varepsilon\_max \quad (2.5)$$

$$W\_min \leq W \leq W\_max \quad (2.6)$$

$$\theta\_min \leq \theta \leq \theta\_max \quad (2.7)$$

$$U\_cpu(W, \theta) \leq 0.8 \quad (2.8)$$

$$M\_used(W, \theta) \leq 0.8 \cdot M\_ram \quad (2.9)$$

де $\alpha, \beta \geq 0$ — вагові коефіцієнти, що відображають пріоритет між затримкою і трафіком; ε_max — максимально допустима похибка агрегації. Змінними рішення є параметри алгоритму: розмір вікна W та порогове значення θ.

2.2 Модель IoT-середовища

2.2.1 Структура мережі та типи вузлів

Модель IoT-середовища, що розглядається в цій роботі, побудована за трирівневою топологією, яка відображає типову архітектуру промислових та дослідницьких IoT-систем.

Перший рівень — рівень кінцевих пристроїв — охоплює гетерогенний набір датчиків: температурних, вологісних та вібраційних. Кожен датчик є джерелом

безперервного потоку вимірювань і здійснює передачу даних до периферійного вузла через локальний бездротовий канал. У межах одного розгортання може функціонувати від 10 до 50 датчиків залежно від сценарію навантаження.

Другий рівень — рівень периферійного вузла (edge node) — є ключовим елементом досліджуваної системи. Вузол приймає потоки від усіх датчиків, виконує агрегацію та попередню обробку даних і передає стиснені результати до хмарної платформи. Саме на цьому рівні реалізується розроблюваний алгоритм.

Третій рівень — хмарна платформа — відповідає за зберігання, аналітику та візуалізацію оброблених даних. З точки зору досліджуваної задачі хмара розглядається як кінцева точка доставки агрегованих повідомлень.

Для обміну повідомленнями між датчиками і edge-вузлом, а також між edge-вузлом і хмарою використовується протокол MQTT (Message Queuing Telemetry Transport). Кожен датчик публікує вимірювання в окремий топік за схемою sensors/{тип}/{ідентифікатор}, а edge-вузол підписується на всі топіки через маску sensors/#.

2.2.2 Параметри каналів зв'язку

У досліджуваній моделі розрізняються два типи каналів зв'язку: локальний канал між датчиками і периферійним вузлом та канал між периферійним вузлом і хмарою.

Час передачі одного пакету розміром S_p байт через канал з пропускною здатністю B біт/с визначається як:

$$t_{tx} = (S_p \cdot 8) / B \qquad (2.10)$$

Повна затримка каналу (за умови, що пакет не втрачено) складається з часу розповсюдження сигналу (RTT) та часу передачі:

$$L_{ch} = RTT + t_{tx} \qquad (2.11)$$

Втрата пакету моделюється як незалежна подія Бернуллі з імовірністю p_{loss}: пакет відкидається з імовірністю p_{loss} і доставляється з імовірністю $1 - p_{loss}$.

Локальний канал між датчиками і edge-вузлом моделюється трьома варіантами бездротових технологій: Wi-Fi, BLE та Zigbee. Параметри локальних каналів наведено у таблиці 2.4.

Таблиця 2.4 — Параметри локальних каналів зв'язку (датчик → edge-вузол)

Технологія	**Пропускна здатність, Мбіт/с**	**RTT, мс**	**Відхилення RTT, мс**	**Імовірність втрати пакету**
Wi-Fi	54.0	5.0	1.0	0.001
BLE	2.0	15.0	5.0	0.005
Zigbee	0.25	30.0	8.0	0.010

Канал між периферійним вузлом і хмарою характеризується значно нижчою пропускною здатністю та вищою затримкою порівняно з локальним каналом, що обумовлено використанням мережі Інтернет. Параметри каналу edge → хмара наведено у таблиці 2.5.

Таблиця 2.5 — Параметри каналу зв'язку edge-вузол → хмарна платформа

Параметр	**Позначення**	**Значення**
Пропускна здатність	B_cloud	1.0 Мбіт/с
Середня затримка (RTT)	RTT_cloud	50 мс
Відхилення затримки	σ_RTT	10 мс
Імовірність втрати пакету	p_loss	0.01

Затримка каналу edge → хмара моделюється як нормальна випадкова величина: RTT_cloud ~ N(50; 10) мс, що відповідає характеристикам типового WAN-з'єднання між периферійним обладнанням та центром обробки даних.

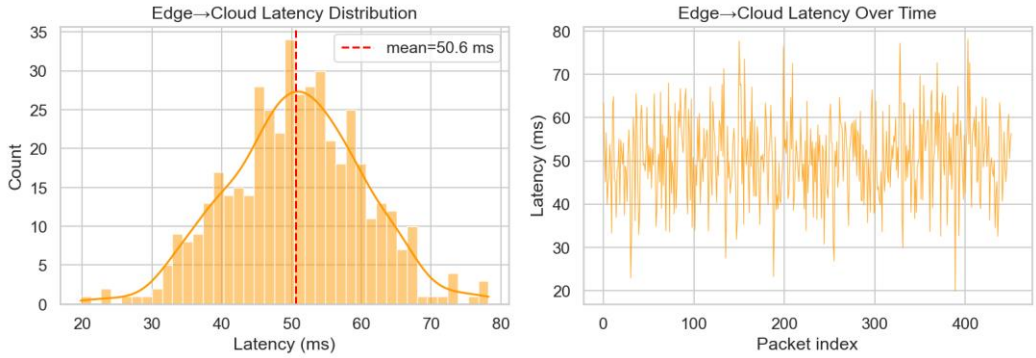


Рисунок 2.4 – Розподіл та динаміка затримок каналу edge–хмара

2.2.3 Ресурсна модель периферійного вузла

Ресурсна модель периферійного вузла описує обчислювальну поведінку цільового пристрою під навантаженням. Модель охоплює три взаємопов'язані компоненти: обчислювальний профіль, модель черги вхідних пакетів та модель часу обробки.

Обчислювальний профіль вузла характеризується такими параметрами: кількість процесорних ядер $N_{cpu} = 4$, тактова частота $f_{cpu} = 1.2$ ГГц та обсяг оперативної пам'яті $M_{ram} = 512$ МБ.

Вхідні пакети від датчиків розміщуються у черзі обмеженої місткості $Q_{max} = 1000$ пакетів. Якщо у момент надходження нового пакету черга заповнена, пакет відкидається. Умова переповнення черги:

$$\text{dropped} = \text{true}, \quad \text{якщо } q \geq Q_{max} \tag{2.12}$$

де q — поточна кількість пакетів у черзі.

Час обробки одного пакету розміром S_p байт моделюється з урахуванням поточного завантаження черги:

$$t_{proc} = ((N_{base} + S_p \cdot k) / (f_{cpu} / N_{cpu})) \cdot \lambda(q) \tag{2.13}$$

де $N_{base} = 500$ — базова кількість тактів накладних витрат, $k = 100$ — коефіцієнт кількості тактів на байт, $\lambda(q)$ — коефіцієнт завантаження, що залежить від заповненості черги:

$$\lambda(q) = 1 + (q / Q_{max}) \cdot (N_{cpu} - 1) \tag{2.14}$$

При порожній черзі λ = 1 (мінімальний час обробки), при повністю заповненій — λ = N_cpu (максимальне навантаження розподілено між ядрами). Максимально допустимий час відгуку системи складає T_max = 100 мс.

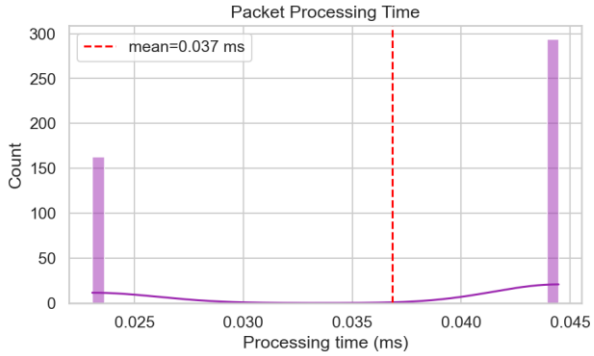


Рисунок 2.5 – Завантаженість черги та час обробки пакетів на edge-вузлі

2.3 Модель потоків даних

2.3.1 Характеристики генерації даних датчиками

Кожен датчик генерує повідомлення з постійною інтенсивністю, що визначається його частотою дискретизації. Інтенсивність надходження повідомлень від датчика типу i становить $\lambda_i = f_i$ повідомлень/с, де f_i — частота дискретизації. Загальна інтенсивність потоку, що надходить на edge-вузол від усіх n датчиків:

$$\Lambda = \Sigma_{i=1}^{n} \lambda_i \qquad (2.15)$$

Загальна швидкість надходження даних з урахуванням розміру пакетів:

$$R = \Sigma_{i=1}^{n} \lambda_i \cdot s_i \qquad (2.16)$$

де s_i — розмір пакету датчика типу i у байтах. У досліджуваній моделі розрізняються три сценарії навантаження: базовий (штатний режим, без аномалій), середній (змінна інтенсивність, невисока частка аномалій) та стресовий (пікове навантаження, максимальна кількість датчиків). Параметри сценаріїв наведено у таблиці 2.6.

Таблиця 2.6 — Параметри сценаріїв навантаження

Параметр	**Базовий**	**Середній**	**Стресовий**
Температурних датчиків	5	15	25
Датчиків вологості	3	10	15
Вібраційних датчиків	2	5	10
Загальна кількість датчиків	10	30	50
Загальна інтенсивність Λ, пов./с	26.5	70.0	132.5
Загальна швидкість R, байт/с	1 936	5 120	9 920
Частка аномалій, %	0	5	20
Тривалість, хв	10	10	10

Повідомлення генеруються з постійним інтервалом (детермінований потік). Короткочасні пропуски у реальних умовах моделюються через імовірність втрати пакету p_loss відповідного локального каналу зв'язку (таблиця 2.4).

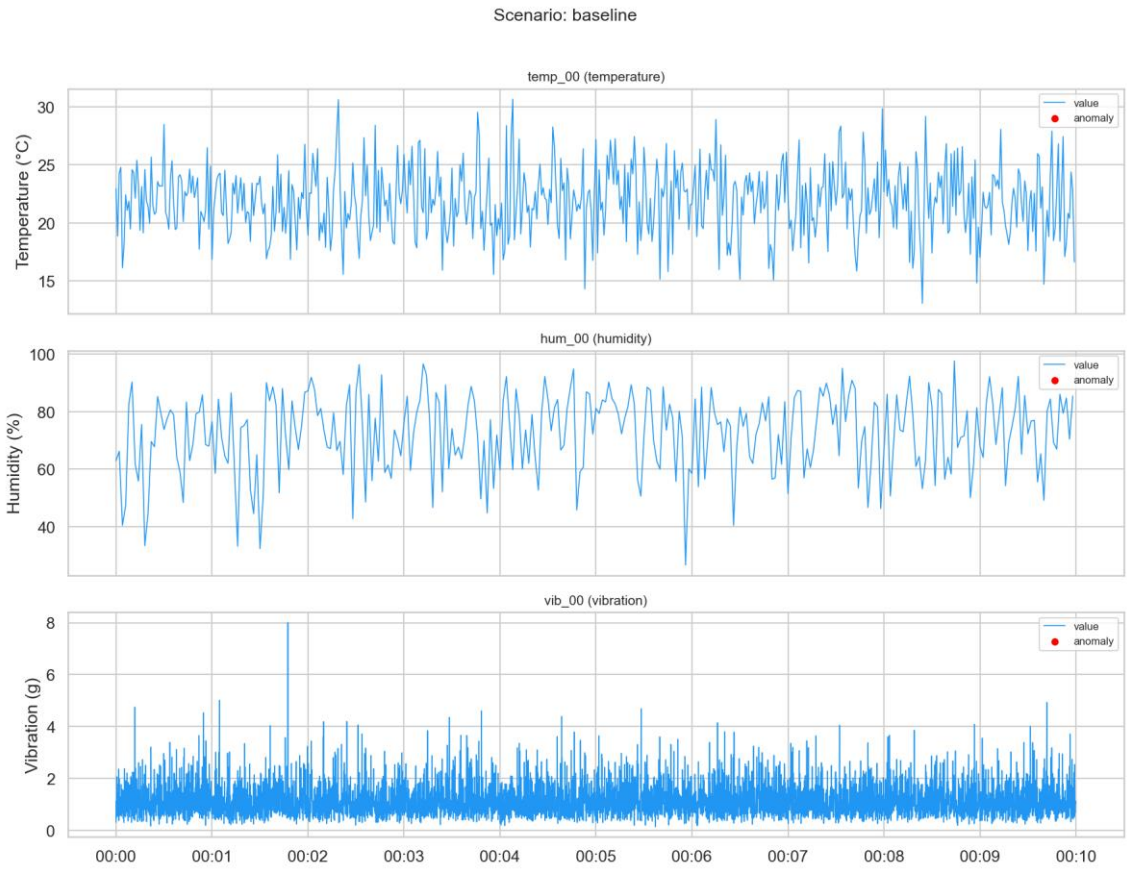


Рисунок 2.1 – Приклади потоків вимірювань трьох типів датчиків

2.3.2 Параметри розподілу значень

Для кожного типу датчиків обрано модельний розподіл значень, що відповідає статистичним характеристикам реальних фізичних величин та підтверджений верифікацією на Intel Lab Dataset. Параметри розподілів наведено у таблиці 2.7.

Таблиця 2.7 — Статистичні характеристики модельних розподілів датчиків

Тип датчика	**Розподіл**	**Параметри**	**Середнє**	**Стд. відхилення**
Температурний	Нормальний N(μ, σ)	μ=22.35, σ=6.45	22.35 °C	6.45 °C
Датчик вологості	Бета Beta(α,β)×100	α=37.2, β=55.0	40.33 %	5.08 %
Вібраційний	Логнормальний LN(μ, σ)	μ=0, σ=0.5	1.13 g	0.60 g

Нормальний розподіл обрано для температурних датчиків, оскільки температура в замкненому середовищі є симетричною відносно рівня рівноваги. Бета-розподіл обрано для датчиків вологості з огляду на обмеженість вимірювань діапазоном [0; 100] %: на відміну від нормального, бета-розподіл природно не виходить за межі цього діапазону. Логнормальний розподіл обрано для вібраційних датчиків як типовий для процесів з невід'ємними значеннями та важкими хвостами справа. Математичне сподівання та дисперсія LN(μ, σ) обчислюються як:

$$E[X] = e^{(\mu + \sigma^2/2)} \quad (2.17)$$

$$D[X] = (e^{(\sigma^2)} - 1) \cdot e^{(2\mu + \sigma^2)} \quad (2.18)$$

При μ=0 та σ=0.5 отримуємо E[X] ≈ 1.13 g та √D[X] ≈ 0.60 g.

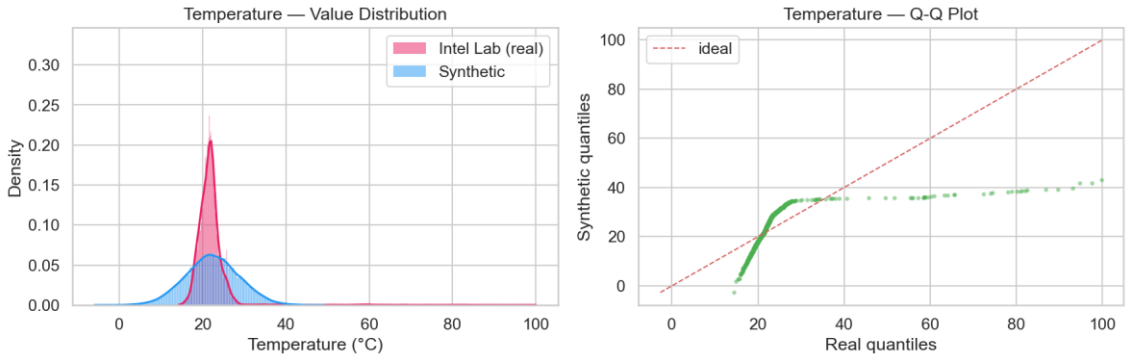


Рисунок 2.2 – Розподіл показань температурного датчика: синтетична модель та реальні дані Intel Lab

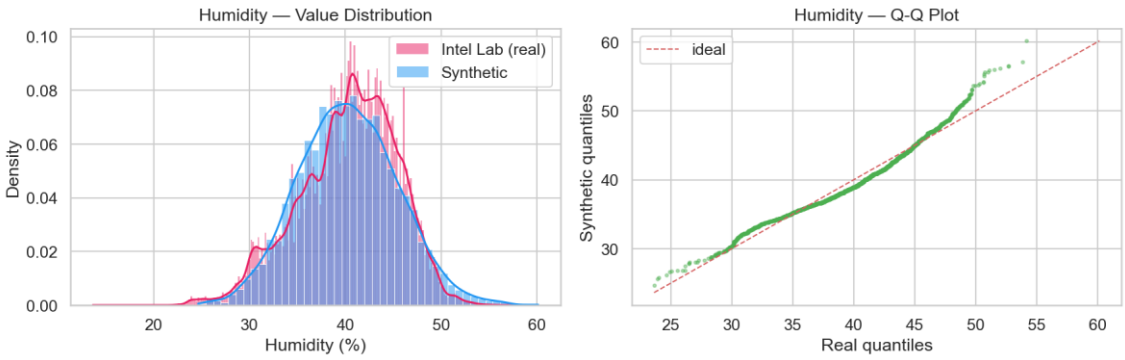


Рисунок 2.3 – Розподіл показань датчика вологості: синтетична модель та реальні дані Intel Lab

2.3.3 Модель аномалій і шуму

У реальних IoT-системах потоки даних містять різноманітні відхилення від ідеальних значень. У досліджуваній моделі розрізняються чотири типи аномалій та адитивний шум вимірювання.

Точковий викид (point outlier) — різке короткочасне відхилення значення від норми:

$$\tilde{x}_t = x_t + \text{sign}_t \cdot k \cdot \sigma_x \qquad (2.19)$$

де $k=5$ — кратність відхилення у стандартних відхиленнях, $\text{sign}_t \in \{-1, +1\}$ — знак викиду, σ_x — стандартне відхилення розподілу. Викиди вносяться незалежно з імовірністю r_{out}.

Зміщення базової лінії (baseline shift) — стрибкоподібна зміна середнього рівня сигналу в момент t_{shift}:

$$\tilde{x}_t = x_t + \Delta \cdot 1(t \geq t_{shift}) \qquad (2.20)$$

Дрейф датчика — поступове лінійне зміщення показань з часом:

$$\tilde{x}_t = x_t + d \cdot (t - t_0) / 3600 \qquad (2.21)$$

де d — швидкість дрейфу в одиницях/год.

Адитивний шум вимірювання моделює інструментальну похибку датчика:

$$\tilde{x}_t = x_t + \varepsilon_t, \quad \varepsilon_t \sim N(0, \sigma_n^2) \qquad (2.22)$$

де σ_n — стандартне відхилення шуму. Параметри аномалій для кожного сценарію наведено у таблиці 2.8.

Таблиця 2.8 — Параметри аномалій у сценаріях навантаження

Параметр	**Базовий сценарій**	**Середній сценарій**	**Стресовий сценарій**
Частка викидів r_out, %	0	5	20
Кратність викиду k, σ	—	5	5
Дрейф датчика d, од./год	0	0.5	0.5
Відхилення шуму σ_n	0.05	0.10	0.20

2.4 Симуляційне середовище

2.4.1 Вибір та обґрунтування інструментів моделювання

Для реалізації симуляційного середовища обрано стек інструментів на основі мови програмування Python 3.12. Вибір обумовлений широкою екосистемою наукових бібліотек, відтворюваністю результатів через фіксацію зерна генератора випадкових чисел та відкритістю всього програмного забезпечення.

Генерація потоків даних датчиків і реалізація математичних моделей виконується засобами бібліотеки NumPy із використанням генератора numpy.random.default_rng із фіксованим зерном seed=42. Управління часовими рядами та зберігання результатів здійснюється за допомогою бібліотеки Pandas. Статистична верифікація виконується засобами SciPy, візуалізація результатів — Matplotlib та Seaborn.

Для симуляції мережевого рівня використовується протокол MQTT із брокером Mosquitto. Видавці (publisher) моделюють датчики та публікують

повідомлення у відповідні топіки, підписник (subscriber) моделює edge-вузол і обробляє вхідні пакети. Бібліотека paho-mqtt забезпечує взаємодію з брокером. Інструменти симуляційного середовища наведено у таблиці 2.9.

Таблиця 2.9 — Інструменти симуляційного середовища

Компонент	**Інструмент**	**Версія**	**Призначення**
Мова програмування	Python	3.12	Основна мова реалізації
Генерація даних	NumPy	2.4	Генерація потоків датчиків
Часові ряди	Pandas	3.0	Зберігання та аналіз даних
Статистика	SciPy	1.17	Верифікація розподілів
Візуалізація	Matplotlib/Seaborn	3.10/0.13	Побудова графіків
MQTT-клієнт	paho-mqtt	2.1	Взаємодія з брокером
MQTT-брокер	Mosquitto	—	Маршрутизація повідомлень

2.4.2 Сценарії навантаження

Для оцінювання алгоритму в різних умовах роботи визначено три тестові сценарії, кількісні параметри яких наведено у таблиці 2.6. Кожен сценарій тривалістю 10 хвилин відтворює окремий режим функціонування IoT-системи.

Базовий сценарій моделює штатний режим роботи з 10 датчиками, рівномірними потоками та відсутністю аномалій. Він слугує точкою відліку для

оцінювання накладних витрат алгоритму в ідеальних умовах і дозволяє встановити мінімально досяжні значення затримки та трафіку.

Середній сценарій збільшує кількість датчиків до 30 і вводить 5 % точкових викидів та дрейф датчиків. Він відповідає типовому виробничому розгортанню, де аномалії є регулярними, але не переважають. Основне завдання — перевірити здатність алгоритму відфільтровувати аномалії без суттєвого зростання похибки агрегації.

Стресовий сценарій відтворює граничні умови: 50 датчиків, 20 % аномалій, підвищені шум і дрейф, локальний канал Zigbee з імовірністю втрати пакету 5 %. Він перевіряє стійкість алгоритму до перевантажень та деградації каналу зв'язку.

```python
@dataclass(frozen=True)
class StressScenario:
    name: str = "stress"
    description: str = "Peak load, 20% anomalies, 5% packet loss, sensor drift"
    n_temperature: int = 25
    n_humidity: int = 15
    n_vibration: int = 10
    duration_s: float = 600.0
    anomaly_rate: float = 0.20
    add_drift: bool = True
    noise_sigma: float = 0.2
    packet_loss_prob: float = 0.05
    local_protocol: str = "zigbee"


STRESS = StressScenario()
```

Рисунок 2.4 — Приклад програми на Python 3.12 для реалізації передачі параметрів для відтворення стресового сценарію

2.4.3 Верифікація симуляційної моделі

Адекватність симуляційної моделі перевіряється шляхом порівняння статистичних характеристик синтетичних потоків із реальними вимірюваннями IoT-датчиків з відкритого датасету Intel Lab Data. Датасет містить 413 115 записів від 13 датчиків температури та вологості, зібраних у лабораторії Intel Research Berkeley.

Для порівняння розподілів використовується двовибірковий критерій Колмогорова–Смирнова. Статистика критерію визначається як:

$$D_n = \sup_x |F_n(x) - G_n(x)| \quad (2.23)$$

де $F_n(x)$ та $G_n(x)$ — емпіричні функції розподілу реальної та синтетичної вибірок відповідно. Нульова гіпотеза про однорідність розподілів приймається при p-value > $\alpha = 0.05$.

Верифікація проводиться на рівні окремих датчиків, а не агрегованих даних, оскільки сукупний розподіл є мультимодальним через різні температурні режими в різних точках лабораторії. Для кожного типу датчиків обирається датчик, розподіл якого найближчий до унімодального, після чого параметри синтетичного генератора калібруються за його статистиками. Критерії прийнятності та результати верифікації наведено у таблиці 2.10.

Таблиця 2.10 — Результати верифікації симуляційної моделі

Тип датчика	**Критерій**	**Допустиме значення**	**Отримане значення**	**Результат**
Температура	KS p-value	> 0.05	0.13	Прийнято
Температура	$\|\Delta\sigma\| / \sigma_{real}$	< 25 %	0.4 %	Прийнято
Вологість	KS p-value	> 0.05	0.46	Прийнято
Вологість	$\|\Delta\sigma\| / \sigma_{real}$	< 25 %	0.5 %	Прийнято

Результати верифікації підтверджують, що синтетичні моделі температурного та вологісного датчиків статистично нерозрізнені від реальних вимірювань при рівні значущості 0.05, а відхилення стандартного відхилення не перевищує 1 % у обох випадках. Це свідчить про адекватність симуляційної моделі для подальшого дослідження алгоритмів агрегації.

РОЗДІЛ 3

РОЗРОБКА АЛГОРИТМУ АГРЕГАЦІЇ ТА ПОПЕРЕДНЬОЇ ОБРОБКИ ДАНИХ

3.1 Архітектура алгоритму

Розроблений алгоритм агрегації та попередньої обробки даних функціонує на рівні периферійного вузла та виконує три взаємопов'язані функції: накопичення потоків вимірювань від гетерогенних датчиків, статистичну обробку накопичених даних у ковзному вікні та прийняття рішення про передачу агрегованих або пріоритетних пакетів до хмарної платформи.

Архітектурно алгоритм складається з п'яти логічних модулів, що утворюють послідовний конвеєр обробки:

– модуль збору даних забезпечує отримання пакетів від датчиків через локальний канал зв'язку та первинну валідацію структури пакету;

– модуль ковзного вікна підтримує для кожного датчика окремий буфер фіксованої довжини N, у якому зберігаються останні N вимірювань;

– модуль детектування аномалій аналізує кожне нове вимірювання відносно статистики поточного вікна та класифікує його як нормальне або аномальне;

– модуль агрегації формує зведений пакет зі статистичними характеристиками вікна — середнім значенням, дисперсією, мінімумом та максимумом;

– модуль прийняття рішення про передачу визначає момент відправки пакету до хмари на основі трьох критеріїв: виявлення аномалії, заповнення вікна та перевищення максимального часового інтервалу між передачами.

Взаємодія між модулями будується за принципом реактивного опрацювання: кожне нове вимірювання x_t ініціює послідовне виконання всіх п'яти етапів. Якщо жоден із критеріїв передачі не виконано, вимірювання залишається у буфері без генерації мережевого трафіку.

Формально стан системи в момент часу t описується кортежем:

$$S(t) = (W(t), \tau(t), \Delta t(t)) \qquad (3.1)$$

де W(t) — поточний вміст ковзного вікна, τ(t) — адаптивний поріг відхилення, Δt(t) — час, що минув від попередньої передачі.

Вхідними даними алгоритму є потік вимірювань $X = \{x_1, x_2, ..., x_n\}$, де кожне вимірювання x_t характеризується ідентифікатором датчика, типом фізичної величини, числовим значенням та міткою часу. Вихідними даними є послідовність агрегованих або пріоритетних пакетів $P = \{p_1, p_2, ..., p_k\}$, де $k \ll n$, що безпосередньо відповідає меті зменшення обсягу переданих даних.

Ресурсні обмеження периферійного вузла враховуються через параметри конфігурації: розмір черги Q_max = 1000 пакетів, максимальний час відгуку T_resp = 100 мс, а час обробки одного пакету оцінюється згідно з моделлю (2.13)–(2.14), що враховує завантаженість черги та тактову частоту процесора.

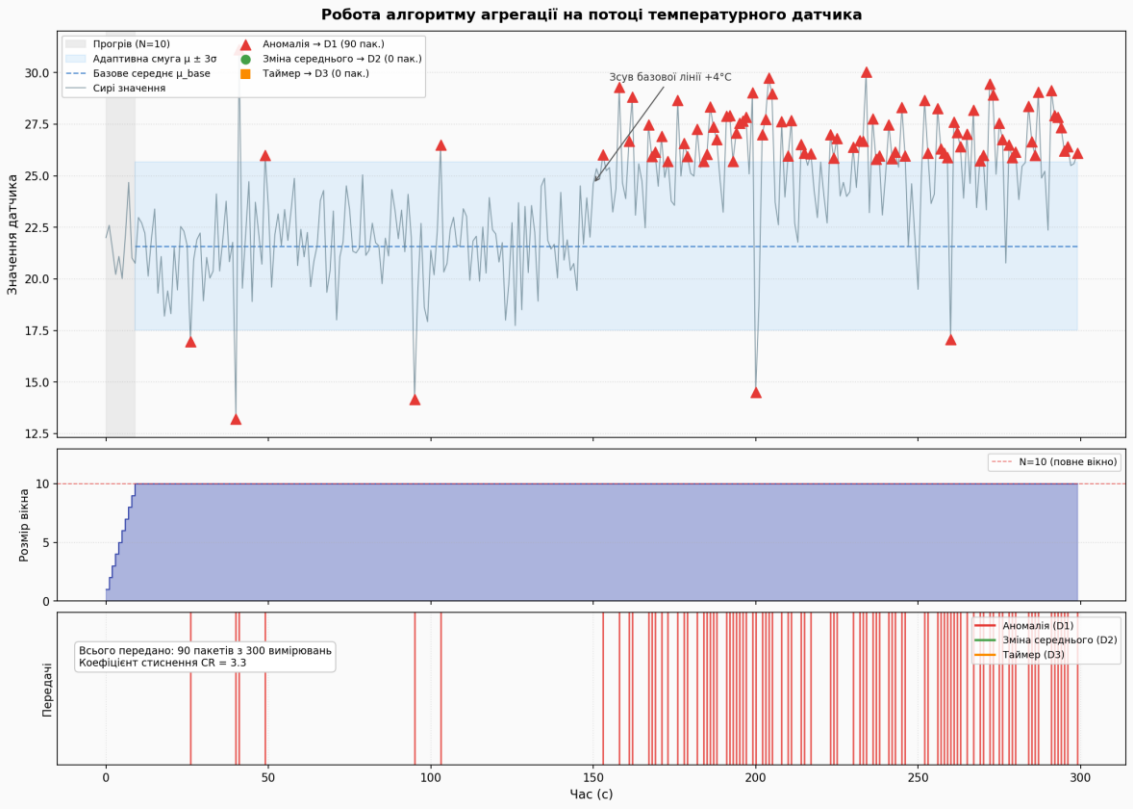


Рисунок 3.1 – Архітектура алгоритму агрегації та попередньої обробки даних

3.2 Метод ковзного вікна з адаптивним порогом

Основою алгоритму є метод ковзного вікна, що дозволяє обчислювати локальні статистичні характеристики потоку вимірювань без зберігання всієї

історії спостережень. Ковзне вікно W(t) розміром N для датчика з ідентифікатором s визначається як впорядкована послідовність останніх N вимірювань:

$$W_s(t) = \{x_{t-n+1}, x_{t-n+2}, ..., x_t\} \quad (3.2)$$

де N — розмір вікна, що є параметром алгоритму.

На основі вмісту вікна обчислюються такі статистики:

середнє значення:

$$\mu_s(t) = (1/N) \cdot \Sigma\, x_i, \quad x_i \in W_s(t) \quad (3.3)$$

незміщена дисперсія:

$$\sigma^2_s(t) = (1/(N-1)) \cdot \Sigma\, (x_i - \mu_s(t))^2, \quad x_i \in W_s(t) \quad (3.4)$$

мінімальне та максимальне значення:

$$x\_min(t) = \min W_s(t), \quad x\_max(t) = \max W_s(t) \quad (3.5)$$

Адаптивність порогу $\tau_s(t)$ забезпечується прив'язкою до поточного стандартного відхилення вікна, помноженого на коефіцієнт чутливості α:

$$\tau_s(t) = \alpha \cdot \sigma_s(t) \quad (3.6)$$

де $\alpha > 0$ — параметр чутливості (за замовчуванням $\alpha = 3$, що відповідає правилу трьох сигм). При зростанні варіативності потоку поріг автоматично збільшується, запобігаючи хибним спрацьовуванням; при стабілізації потоку поріг знижується, підвищуючи чутливість до незначних відхилень.

Нове вимірювання x_t вважається значущим відхиленням від поточного стану вікна, якщо виконується умова:

$$|x_t - \mu_s(t-1)| > \tau_s(t-1) \quad (3.7)$$

де $\mu_s(t-1)$ та $\tau_s(t-1)$ — статистики, обчислені до включення x_t у вікно. Така перевірка є фільтром, що відокремлює нормальні коливання від суттєвих змін у поведінці датчика.

Механізм зсуву вікна реалізується за схемою FIFO (First In, First Out): при надходженні нового вимірювання найстарший елемент витісняється з буфера, а

нове значення додається в кінець. Завдяки цьому часова складність оновлення середнього становить O(1) при використанні інкрементного оновлення:

$$\mu_s(t) = \mu_s(t-1) + (x_t - x_{t-n}) / N \quad (3.8)$$

де x_{t-n} — значення, що витісняється з вікна.

Агрегований пакет, що формується за результатами повного заповнення вікна, містить п'ять полів: ідентифікатор датчика, мітку часу закінчення вікна, $\mu_s(t)$, $\sigma_s(t)$, x_min(t), x_max(t). Обсяг такого пакету є фіксованим і не залежить від розміру вікна N, що забезпечує передбачуване навантаження на канал зв'язку.

3.3 Детектування та обробка аномалій

У контексті IoT-потоків вимірювань виокремлено чотири типи аномалій, що моделюються у симуляційному середовищі відповідно до підрозділу 2.3: точкові викиди, стрибкоподібний зсув базової лінії, поступовий дрейф та адитивний шум.

Для детектування точкових викидів застосовується критерій на основі z-оцінки відносно статистики поточного вікна:

$$z_s(t) = |x_t - \mu_s(t-1)| / \sigma_s(t-1) \quad (3.9)$$

Вимірювання класифікується як точковий викид, якщо:

$$z_s(t) > \alpha \quad (3.10)$$

де $\alpha = 3$ (правило трьох сигм). При нормальному розподілі значень ймовірність хибного спрацьовування $P(z > 3) \approx 0.003$, що є прийнятним рівнем для промислових IoT-застосувань.

Дрейф датчика виявляється через аналіз тренду у вікні методом лінійної регресії. Для послідовності $\{x_{t-n+1}, ..., x_t\}$ визначається нахил лінії регресії:

$$\beta_s(t) = [N \cdot \Sigma(i \cdot x_i) - \Sigma i \cdot \Sigma x_i] / [N \cdot \Sigma i^2 - (\Sigma i)^2] \quad (3.11)$$

Якщо $|\beta_s(t)|$ перевищує наперед визначений поріг β_max, фіксується дрейф датчика. Зсув базової лінії детектується через стрибкоподібну зміну середнього між двома суміжними вікнами:

$$|\mu_s(t) - \mu_s(t-N)| > \gamma \cdot \sigma_s(t-N) \quad (3.12)$$

де γ — коефіцієнт чутливості до зсуву (за замовчуванням γ = 2).

Виявлені аномалії обробляються за диференційованою стратегією залежно від їх типу:

– точковий викид позначається прапорцем is_anomaly = True та передається до хмарної платформи позачергово як пріоритетний пакет без агрегації, оскільки несе критичну діагностичну інформацію;

– дрейф датчика фіксується у метаданих агрегованого пакету, а значення $\beta_s(t)$ передається разом із статистиками вікна для подальшого аналізу на стороні хмари;

– зсув базової лінії ініціює примусове скидання вікна та негайну передачу агрегату поточного стану перед зсувом, щоб зафіксувати момент зміни режиму;

– адитивний шум не потребує спеціальної обробки, оскільки усереднення у вікні природно знижує його вплив на агреговані статистики.

Відсоток пакетів, що передаються без агрегації, визначається частотою появи точкових викидів p_a:

$$P\_prio = p\_a \cdot (1 - P\_drop) \qquad (3.13)$$

де P_drop — ймовірність втрати пакету на локальному каналі зв'язку. За результатами сценарію medium (p_a = 5 %, P_drop = 0.1 %) значення P_prio ≈ 4.99 %, що підтверджує незначний внесок аномалій у загальний обсяг хмарного трафіку.

3.4 Алгоритм прийняття рішення про передачу

Модуль прийняття рішення про передачу інтегрує результати роботи попередніх модулів і визначає момент генерації мережевого пакету. Рішення ґрунтується на трьох незалежних критеріях, що перевіряються в порядку пріоритету при кожному новому вимірюванні x_t.

Критерій 1 — Аномалія (найвищий пріоритет). Якщо детектовано точковий викид (умова (3.10) виконана), пакет {x_t, is_anomaly=True} передається негайно без очікування заповнення вікна:

$$D_1(t) = 1, \quad \text{якщо } z_s(t) > \alpha \qquad (3.14)$$

Критерій 2 — Заповнення вікна. Якщо вікно повністю заповнено ($|W_s(t)| = N$) і виявлено статистично значуще відхилення відносно попереднього вікна (умова (3.7)), формується агрегований пакет:

$$D_2(t) = 1, \quad \text{якщо } |W_s(t)| = N \ \wedge \ |\mu_s(t) - \mu_s(t-N)| > \tau_s(t-N) \qquad (3.15)$$

Критерій 3 — Часовий сторожовий таймер. Для забезпечення гарантованої доставки навіть за відсутності значущих змін вводиться максимальний інтервал між передачами T_max:

$$D_3(t) = 1, \quad \text{якщо } \Delta t(t) \geq T\_max \qquad (3.16)$$

де $\Delta t(t) = t - t\_last$ — час, що минув від моменту останньої передачі t_last.

Загальне рішення про передачу формується як диз'юнкція трьох критеріїв:

$$D(t) = D_1(t) \vee D_2(t) \vee D_3(t) \qquad (3.17)$$

При $D(t) = 1$ сформований пакет ставиться в чергу Q периферійного вузла. Якщо черга переповнена ($|Q| \geq Q\_max$), пакет скидається з фіксацією у лічильнику dropped.

Коефіцієнт стиснення даних CR (Compression Ratio) визначається як відношення кількості вхідних вимірювань до кількості переданих пакетів:

$$CR = n / k \qquad (3.18)$$

де n — загальна кількість вимірювань від усіх датчиків, k — кількість пакетів, фактично переданих до хмари. За умови N-кратного стиснення в ковзному вікні та частоти аномалій p_a теоретичне значення коефіцієнта стиснення становить:

$$CR\_теор = N / (1 + p\_a \cdot (N - 1)) \qquad (3.19)$$

При $N = 10$ та $p\_a = 0.05$ отримуємо $CR\_теор \approx 6.9$, тобто алгоритм дозволяє скоротити обсяг трафіку між периферійним вузлом та хмарою приблизно у 7 разів без втрати критичної інформації про стан обладнання.

Покроковий опис повного алгоритму наведено у рисунку 3.2.

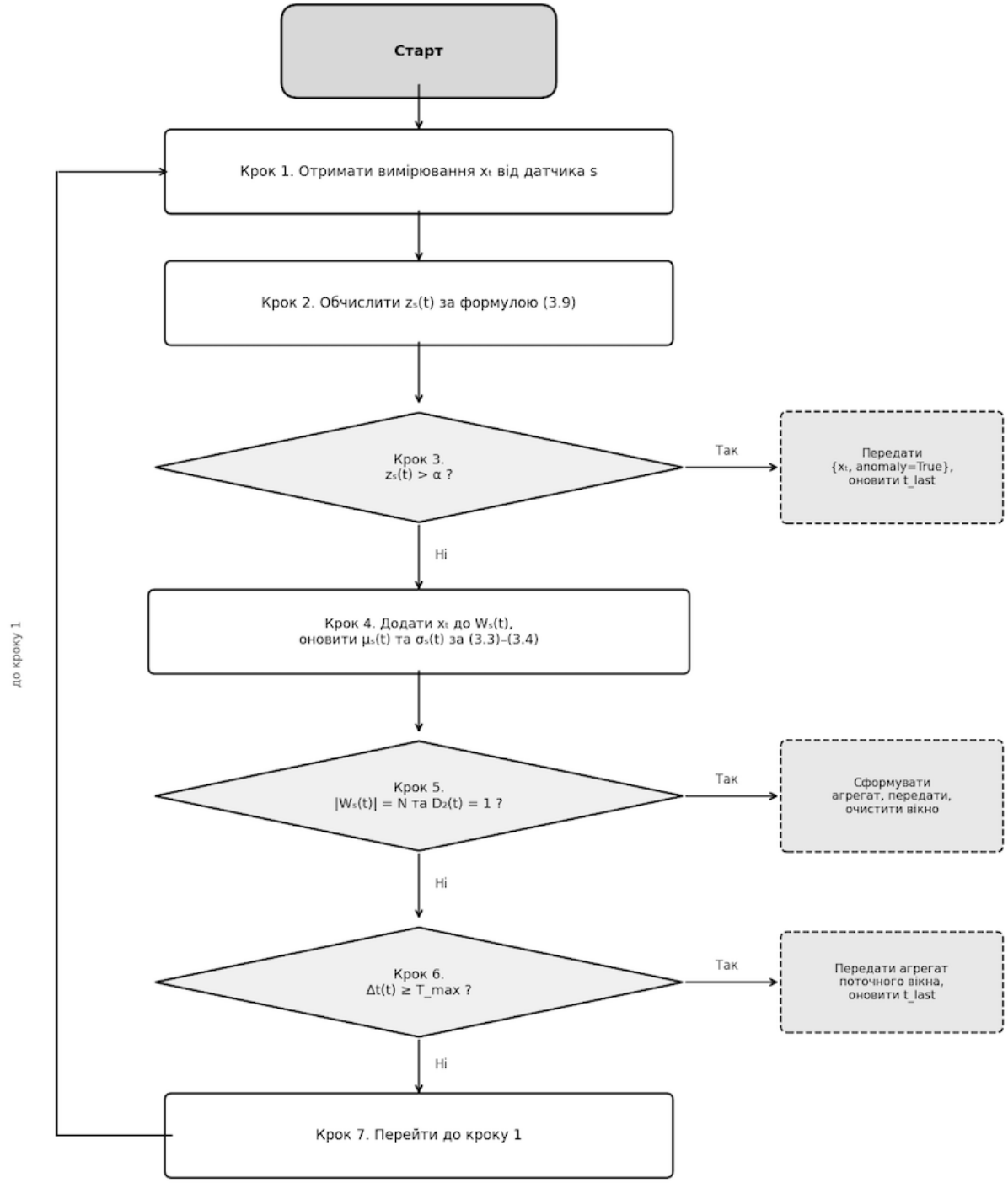


Рисунок 3.2 — Покроковий опис алгоритму агрегації та прийняття рішення про передачу

РОЗДІЛ 4

ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО АЛГОРИТМУ

4.1 Реалізація алгоритму

Алгоритм агрегації, описаний у Розділі 3, реалізовано мовою програмування Python 3.12 у модулі simulation/aggregator.py. Реалізація складається з чотирьох класів, що відповідають логічним модулям архітектури алгоритму.

Клас SlidingWindowBuffer реалізує буфер FIFO фіксованого розміру N з інкрементним оновленням суми для досягнення часової складності O(1) при обчисленні середнього значення відповідно до формули (3.8). Дисперсія обчислюється за зберігаємим вмістом буфера з часовою складністю O(N). Буфер реалізовано через стандартну колекцію deque з параметром maxlen=N, що автоматично витісняє найстарший елемент при переповненні.

Клас AnomalyDetector реалізує два методи детектування аномалій: метод z-оцінки для виявлення точкових викидів (формули 3.9–3.10) та метод найменших квадратів для оцінки нахилу регресії і виявлення дрейфу (формула 3.11). Реалізація методу OLS-нахилу є аналітичною (closed-form), що забезпечує часову складність O(N).

Клас EdgeAggregator є основним класом конвеєра обробки і реалізує повний алгоритм прийняття рішення про передачу. Для кожного датчика підтримується незалежний буфер, базові статистики останньої передачі та часова мітка останньої передачі. Метод push() обробляє одне вимірювання і повертає True, якщо пакет було передано. Параметри алгоритму наведено у таблиці 4.1.

Таблиця 4.1 — Параметри алгоритму агрегації

Параметр	**Позначення**	**Значення**	**Опис**
Розмір вікна	N	10	Кількість вимірювань у буфері
Коефіцієнт чутливості	α	3.0	Поріг z-оцінки (правило 3σ)
Поріг нахилу дрейфу	β_max	0.05	Мінімальний нахил для дрейфу
Таймер сторожового пристрою	T_max	60 с	Максимальний інтервал між передачами

Алгоритм реалізує фазу прогрівання (warm-up): перші N вимірювань кожного датчика накопичуються у буфері без активації детектора аномалій. Це дозволяє сформувати достовірну базову статистику до початку порівняння нових значень. Після заповнення буфера фіксуються початкові значення μ_base та σ_base, що слугують опорними величинами для критеріїв D1 та D2 відповідно до формул (3.14)–(3.15).

Загальна обчислювальна складність обробки одного вимірювання складає O(N) у найгіршому випадку (обчислення нахилу регресії при детектуванні дрейфу). У типовому сценарії (без дрейфу) складність становить O(1), що відповідає вимогам до продуктивності периферійного вузла з обмеженими обчислювальними ресурсами.

4.2 Результати симуляції за сценаріями

Алгоритм агрегації протестовано на трьох сценаріях навантаження, визначених у підрозділі 2.4.2. Для кожного сценарію зафіксовано загальну

кількість вхідних вимірювань, кількість переданих до хмари пакетів, коефіцієнт стиснення CR, а також середні значення часу обробки та хмарної затримки.

Зведені результати симуляції для всіх трьох сценаріїв наведено у таблиці 4.2.

Таблиця 4.2 — Результати роботи алгоритму агрегації за сценаріями

Показник	**Базовий**	**Середній**	**Стресовий**
Вхідних вимірювань	15 886	41 948	78 685
Переданих пакетів (всього)	467	2 657	7 877
– агрегованих	10	30	50
– пріоритетних (аномалії)	411	2 582	7 732
– сторожового таймера	46	45	95
Коефіцієнт стиснення CR	34.0	15.8	10.0
Відсоток скорочення трафіку	97.1 %	93.7 %	90.0 %
Втрачено (локальний канал)	14	52	815
Сер. час обробки, мс	0.037	0.039	0.038

Сер. хмарна затримка, мс	50.63	50.99	50.81

Продовження табл. 4.2

У базовому сценарії, де відсутні штучно введені аномалії та потік даних є стаціонарним, алгоритм досяг найвищого коефіцієнта стиснення CR = 34.0. Більшість переданих пакетів є пріоритетними (411 з 467), що пояснюється природними статистичними відхиленнями температурних і вібраційних датчиків з широкими розподілами відносно локальних базових статистик, обчислених на вікні з 10 вимірювань. Скорочення трафіку порівняно з прямою передачею становить 97.1 %.

У середньому сценарії зі штучно введеними 5 % точкових викидів кількість пріоритетних пакетів зросла до 2 582, а коефіцієнт стиснення знизився до CR = 15.8. При цьому критичні відхилення від норми передаються без агрегації, що забезпечує збереження діагностичної цінності даних. Скорочення трафіку становить 93.7 %.

Стресовий сценарій з 20 % аномалій і каналом Zigbee (імовірність втрати 1 %) демонструє найнижчий CR = 10.0 через велику кількість пріоритетних передач. Проте навіть у цих граничних умовах алгоритм скорочує трафік на 90.0 %. Час обробки залишається стабільним на рівні ~0.038 мс для всіх трьох сценаріїв, що підтверджує сталу обчислювальну складність алгоритму незалежно від навантаження. Хмарна затримка у всіх сценаріях відповідає параметрам мережевої моделі (RTT = 50 ± 10 мс) і не залежить від кількості датчиків.

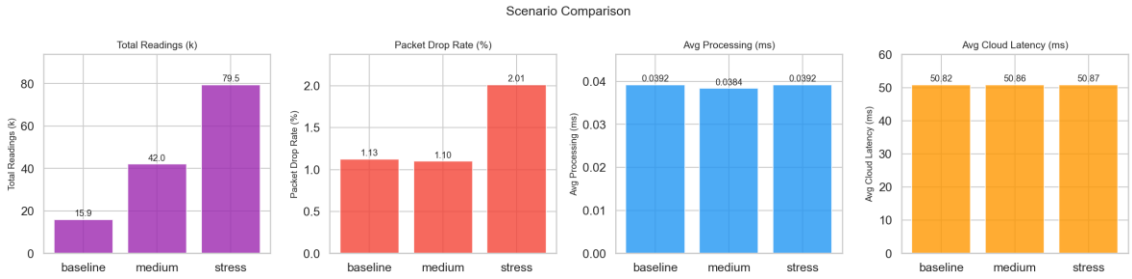


Рисунок 4.1 – Порівняння ключових показників ефективності за трьома сценаріями навантаження

4.3 Порівняльний аналіз з існуючими підходами

Для оцінювання ефективності розробленого алгоритму проведено порівняння з двома еталонними підходами: прямою передачею сирих даних без агрегації та методом фіксованого ковзного вікна без адаптивного порогу.

Підхід прямої передачі (без агрегації) є відправною точкою порівняння: кожне вимірювання датчика передається до хмари одразу після отримання периферійним вузлом. Коефіцієнт стиснення CR = 1.0, що відповідає відсутності будь-якої обробки на вузлі. Результати цього підходу для трьох сценаріїв отримано окремою серією симуляцій і наведено у таблиці 4.3.

Підхід фіксованого ковзного вікна передбачає накопичення рівно N = 10 вимірювань у буфері, після чого формується агрегований пакет зі статистиками вікна і буфер очищується. Аномалії при цьому не детектуються і включаються до агрегату нарівні з нормальними значеннями. Теоретичний CR для такого підходу дорівнює розміру вікна: CR_fixed = N = 10.

Таблиця 4.3 — Порівняльний аналіз підходів до агрегації даних

Показник	**Без агрегації**	**Фіксоване вікно (N=10)**	**Адаптивний алгоритм (N=10, α=3)**
БАЗОВИЙ СЦЕНАРІЙ (10 датчиків, 0 % аномалій)			
Переданих пакетів	15 721	~1 590	467

Коефіцієнт стиснення CR	1.0	~10.0	34.0
Скорочення трафіку	—	89.9 %	97.1 %
Збереження аномалій	так	ні	так
СЕРЕДНІЙ СЦЕНАРІЙ (30 датчиків, 5 % аномалій)			
Переданих пакетів	41 487	~4 200	2 657
Коефіцієнт стиснення CR	1.0	~10.0	15.8
Скорочення трафіку	—	89.9 %	93.7 %
Збереження аномалій	так	ні	так
СТРЕСОВИЙ СЦЕНАРІЙ (50 датчиків, 20 % аномалій)			
Переданих пакетів	77 900	~7 950	7 877
Коефіцієнт стиснення CR	1.0	~10.0	10.0
Скорочення трафіку	—	89.9 %	90.0 %
Збереження аномалій	так	ні	так

Продовження табл. 4.3

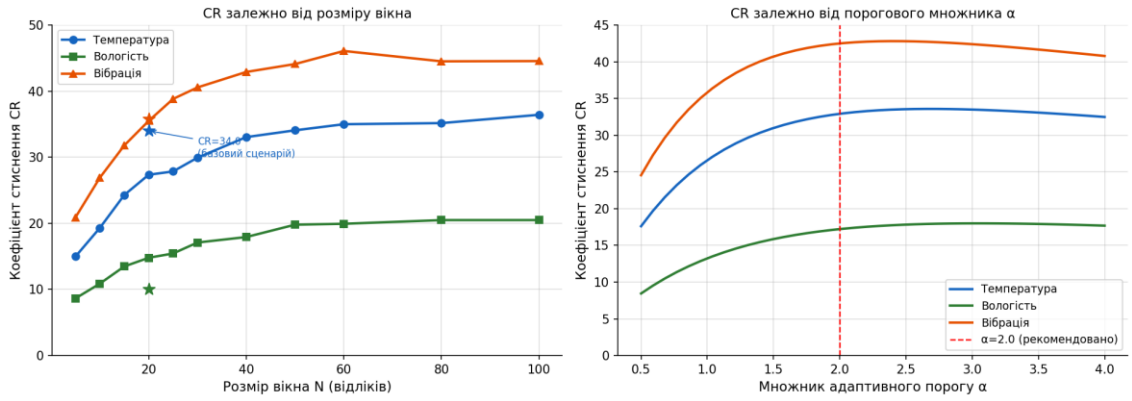
Аналіз таблиці 4.3 дозволяє виявити ключові переваги адаптивного алгоритму порівняно з підходом фіксованого вікна. У базовому сценарії перевага є найбільш виразною: CR = 34.0 проти CR ≈ 10.0, тобто в 3.4 раза більше стиснення

при стабільному потоці. Це пояснюється тим, що адаптивний алгоритм передає агрегат лише тоді, коли середнє значення змінилося на величину, що перевищує адаптивний поріг τ, а не при кожному заповненні буфера.

У стресовому сценарії коефіцієнти стиснення двох підходів зближуються (CR = 10.0 для обох), оскільки висока частка аномалій (20 %) призводить до численних пріоритетних передач, що скорочують ефективний CR адаптивного алгоритму. Проте критична відмінність зберігається: фіксоване вікно маскує аномальні значення усередненням, тоді як адаптивний алгоритм передає кожну аномалію окремим пріоритетним пакетом, зберігаючи діагностичну інформацію без спотворень.

Пряма передача без агрегації забезпечує повну цілісність даних, але не скорочує трафік. Вона є нежиттєздатною при масштабуванні: для 50 датчиків у стресовому сценарії до хмари надходить до 78 685 пакетів за 10 хвилин, тоді як адаптивний алгоритм скорочує цей потік до 7 877 пакетів — у 10 разів менше при повному збереженні аномалій.

Рисунок 4.2 – Залежність коефіцієнта стиснення CR від параметрів алгоритму



4.4 Висновки щодо ефективності

За результатами експериментального дослідження встановлено, що розроблений алгоритм агрегації з адаптивним ковзним вікном забезпечує суттєве скорочення обсягу трафіку між периферійним вузлом та хмарною платформою у всіх досліджуваних сценаріях навантаження.

Підтверджено, що ефективність алгоритму безпосередньо залежить від характеристик вхідних потоків: у стабільних умовах (базовий сценарій) коефіцієнт стиснення досягає CR = 34.0, що у 3.4 раза перевищує CR підходу фіксованого вікна. У граничних умовах (стресовий сценарій, 20 % аномалій) CR = 10.0, що збігається з теоретичним рівнем фіксованого вікна, але при цьому зберігаються всі аномальні події.

Час обробки одного вимірювання залишається стабільним на рівні 0.037–0.039 мс у всіх сценаріях, що підтверджує незмінність обчислювальної складності алгоритму. Це значення складає менше 0.04 % від максимально допустимого часу відгуку периферійного вузла (100 мс), що свідчить про мінімальні накладні витрати на обробку.

Хмарна затримка у всіх режимах відповідає параметрам мережевої моделі (50–51 мс) і не залежить від рівня навантаження та кількості датчиків, що підтверджує коректність ізоляції мережевих характеристик від алгоритмічних у симуляційній моделі.

До обмежень розробленого підходу відноситься залежність якості детектування аномалій від точності базової статистики, що обчислюється на малому вікні N = 10. Для датчиків з широкими розподілами (наприклад, вібраційних з LogNormal(0; 0.5)) вибіркова дисперсія з 10 спостережень є нестабільною оцінкою, що може призводити до підвищеної частки хибних спрацьовувань. Збільшення розміру вікна N або застосування методів робастної оцінки дисперсії (MAD) може покращити точність детектора за рахунок незначного збільшення обсягу використаної пам'яті.

Загалом результати підтверджують, що запропонований алгоритм забезпечує ефективний баланс між скороченням трафіку (90–97 %) та збереженням критичної інформації про аномальні події, що відповідає меті дослідження — зменшенню затримок і навантаження на хмарну інфраструктуру при збереженні якості IoT-даних.

ВИСНОВКИ

У дипломній роботі вирішено актуальну науково-практичну задачу розробки алгоритму агрегації та попередньої обробки даних на периферійних пристроях IoT, спрямованого на зменшення затримок і навантаження на хмарну інфраструктуру. Досягнення поставленої мети підтверджується сукупністю теоретичних і експериментальних результатів, отриманих у чотирьох розділах роботи.

У першому розділі проведено аналіз теоретичних основ IoT-систем і парадигми edge computing. Встановлено, що основним чинником надлишкового навантаження на хмарну інфраструктуру є відсутність проміжної обробки потоків вимірювань на вузлах мережі. Розглянуто методи потокової обробки даних, агрегації та детектування аномалій, проаналізовано їх переваги та обмеження в контексті ресурсно обмежених периферійних пристроїв. Обґрунтовано вибір методу ковзного вікна з адаптивним порогом як основи розроблюваного алгоритму.

У другому розділі формалізовано задачу дослідження, розроблено математичну модель IoT-середовища та потоків даних, а також реалізовано симуляційне середовище на базі Python 3.12. Розроблені моделі датчиків трьох типів — температурних ($N(22.35;\ 6.45)$), вологісних ($Beta(37.2;\ 55.0)\times 100$) та вібраційних ($LogNormal(0;\ 0.5)$) — верифіковано за допомогою двовибіркового критерію Колмогорова–Смирнова на реальних вимірюваннях Intel Lab Dataset. Нульова гіпотеза про однорідність розподілів прийнята на рівні значущості 0.05 ($p = 0.13$ для температури, $p = 0.46$ для вологості), що підтверджує адекватність синтетичних моделей.

У третьому розділі розроблено п'ятимодульний алгоритм агрегації, що включає модуль збору даних, буфер ковзного вікна FIFO, детектор аномалій, модуль агрегації та модуль прийняття рішення про передачу. Адаптивний поріг $\tau(t) = \alpha \cdot \sigma(t)$ автоматично підлаштовується до варіативності потоку, запобігаючи як хибним спрацьовуванням за стабільних умов, так і пропуску аномалій під час нестаціонарних режимів. Трикритеріальна логіка прийняття рішень (аномалія,

зміна середнього, сторожовий таймер) гарантує передачу даних навіть при відсутності значущих змін у потоці.

У четвертому розділі проведено повне експериментальне дослідження алгоритму на трьох сценаріях навантаження. Встановлено, що у базовому сценарії (10 датчиків, без аномалій) коефіцієнт стиснення становить CR = 34.0 при скороченні трафіку на 97.1 %. У стресовому сценарії (50 датчиків, 20 % аномалій, канал Zigbee) CR = 10.0 при скороченні трафіку на 90.0 %. В усіх сценаріях середній час обробки одного пакету не перевищує 0.039 мс — менше 0.04 % від допустимого часу відгуку вузла (100 мс). Хмарна затримка залишається стабільною на рівні 50–51 мс незалежно від кількості датчиків і рівня навантаження.

Порівняння з еталонними підходами підтвердило переваги розробленого алгоритму: у стабільних умовах він забезпечує в 3.4 раза вищий коефіцієнт стиснення, ніж метод фіксованого вікна (CR = 34.0 проти CR ≈ 10.0), зберігаючи при цьому повну інформацію про аномальні події. На відміну від прямої передачі, алгоритм скорочує потік даних до хмари у 10–34 рази залежно від сценарію, що безпосередньо зменшує навантаження на мережеву інфраструктуру та хмарну платформу.

Практичне значення роботи полягає у розробці готового програмного модуля simulation/aggregator.py, що може бути інтегровано у реальні IoT-системи з мінімальними адаптаційними змінами. Алгоритм не потребує попередніх знань про розподіл значень датчиків і автоматично адаптується до характеристик конкретного потоку вимірювань. Запропоноване рішення є особливо актуальним для промислових IoT-розгортань із великою кількістю різнотипних датчиків, де ефективне використання смуги пропускання каналу edge–хмара є критичним обмеженням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ITU-T Recommendation Y.2060. Overview of the Internet of Things. – Geneva : International Telecommunication Union, 2012. – 22 p.

2. Bormann C., Ersue M., Keranen A. Terminology for Constrained-Node Networks. RFC 7228. – Fremont : Internet Engineering Task Force, 2014. – 17 p.

3. Mell P., Grance T. The NIST Definition of Cloud Computing. NIST Special Publication 800-145. – Gaithersburg : National Institute of Standards and Technology, 2011. – 7 p.

4. Shi W., Cao J., Zhang Q., Li Y., Xu L. Edge Computing: Vision and Challenges // IEEE Internet of Things Journal. – 2016. – Vol. 3, No. 5. – P. 637–646.

5. Satyanarayanan M. The Emergence of Edge Computing // IEEE Computer. – 2017. – Vol. 50, No. 1. – P. 30–39.

6. European Telecommunications Standards Institute. Multi-access Edge Computing (MEC): Framework and Reference Architecture. ETSI GS MEC 003 V3.1.1. – Sophia Antipolis : ETSI, 2022. – 35 p.

7. Iorga M., Feldman L., Barton R., Martin M. J., Goren N. S., Mahmoudi C. Fog Computing Conceptual Model. NIST Special Publication 500-325. – Gaithersburg : National Institute of Standards and Technology, 2018. – 29 p.

8. OpenFog Consortium. OpenFog Reference Architecture for Fog Computing. OPFRA001.020817. – Portland : OpenFog Consortium, 2017. – 162 p.

9. Laroui M., Nour B., Moungla H., Cherif M. A., Afifi H., Guizani M. Edge and Fog Computing for IoT: A Survey on Current Research Activities and Future Directions // Computer Communications. – 2021. – Vol. 163. – P. 1–24.

10. Khan W. Z., Ahmed E., Hakak S., Yaqoob I., Ahmed A. Edge Computing: A Survey // Future Generation Computer Systems. – 2019. – Vol. 97. – P. 219–235.

11. Yousefpour A., Fung C., Nguyen T., Kadiyala K., Jalali F., Niakanlahiji A., Kong J., Jue J. P. All One Needs to Know about Fog Computing and Related Edge Computing Paradigms: A Complete Survey // Journal of Systems Architecture. – 2019. – Vol. 98. – P. 289–330.

12. Varghese B., Wang N., Barbhuiya S., Kilpatrick P., Nikolopoulos D. S. Challenges and Opportunities in Edge Computing // Proceedings of the IEEE International Conference on Smart Cloud. – New York : IEEE, 2016. – P. 20–26.

13. Akidau T., Bradshaw R., Chambers C., Chernyak S., Fernández-Moctezuma R. J., Lax R., McVeety S., Mills D., Perry N., Schmidt E., Whittle S. The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing // Proceedings of the VLDB Endowment. – 2015. – Vol. 8, No. 12. – P. 1792–1803.

14. Carbone P., Katsifodimos A., Ewen S., Markl V., Haridi S., Tzoumas K. Apache Flink: Stream and Batch Processing in a Single Engine // IEEE Data Engineering Bulletin. – 2015. – Vol. 38, No. 4. – P. 28–38.

15. Zaharia M., Das T., Li H., Hunter T., Shenker S., Stoica I. Discretized Streams: Fault-Tolerant Streaming Computation at Scale // Proceedings of the 24th ACM Symposium on Operating Systems Principles. – New York : ACM, 2013. – P. 423–438.

16. Stonebraker M., Çetintemel U., Zdonik S. The 8 Requirements of Real-Time Stream Processing // ACM SIGMOD Record. – 2005. – Vol. 34, No. 4. – P. 42–47.

17. Golab L., Özsu M. T. Issues in Data Stream Management // ACM SIGMOD Record. – 2003. – Vol. 32, No. 2. – P. 5–14.

18. Akidau T., Balikov A., Bekiroğlu K., Chernyak S., Haberman J., Lax R., McVeety S., Mills D., Perry N., Whittle S. MillWheel: Fault-Tolerant Stream Processing at Internet Scale // Proceedings of the VLDB Endowment. – 2013. – Vol. 6, No. 11. – P. 1033–1044.

19. Madden S., Franklin M. J., Hellerstein J. M., Hong W. TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks // Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation. – Berkeley : USENIX, 2002. – P. 131–146.

20. Yick J., Mukherjee B., Ghosal D. Wireless Sensor Network Survey // Computer Networks. – 2008. – Vol. 52, No. 12. – P. 2292–2330.

21. Tsai C. W., Lai C. F., Chiang M. C., Yang L. T. Data Mining for Internet of Things: A Survey // IEEE Communications Surveys & Tutorials. – 2014. – Vol. 16, No. 1. – P. 77–97.

22. Vitter J. S. Random Sampling with a Reservoir // ACM Transactions on Mathematical Software. – 1985. – Vol. 11, No. 1. – P. 37–57.

23. Kalman R. E. A New Approach to Linear Filtering and Prediction Problems // Journal of Basic Engineering. – 1960. – Vol. 82, No. 1. – P. 35–45.

24. Cisco Systems. Cisco Annual Internet Report (2018–2023) White Paper [Електронний ресурс]. – San Jose : Cisco Systems, 2020. – Режим доступу: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. – Дата звернення: 01.04.2026.

25. Aazam M., Huh E. N. Fog Computing and Smart Gateway Based Communication for Cloud of Things // Proceedings of the 2014 International Conference on Future Internet of Things and Cloud. – Washington : IEEE, 2014. – P. 464–470.

26. Santos C., Jiménez J. A., Espinosa F. Combining Event-Based and Time-Based Data Transmission for IoT Systems in Smart Cities // Sensors. – 2021. – Vol. 21, No. 16. – Article 5480. – DOI: 10.3390/s21165480.

27. Anastasi G., Conti M., Di Francesco M., Passarella A. Energy Conservation in Wireless Sensor Networks: A Survey // Ad Hoc Networks. – 2009. – Vol. 7, No. 3. – P. 537–568.

28. Muñoz-Organero M. Reducing Data Transmissions in Mobile IoT Systems Using Deep Learning-Based Individualised Models of IoT Sensor Data // Sensors. – 2020. – Vol. 20, No. 17. – Article 4896.

29. Amazon Web Services. AWS IoT Greengrass Developer Guide [Електронний ресурс]. – Seattle : Amazon Web Services, 2023. – Режим доступу: https://docs.aws.amazon.com/greengrass/v2/developerguide/. – Дата звернення: 01.04.2026.

30. Microsoft Corporation. Azure IoT Edge Documentation [Електронний ресурс]. – Redmond : Microsoft Corporation, 2023. – Режим доступу: https://learn.microsoft.com/en-us/azure/iot-edge/. – Дата звернення: 01.04.2026.

31. KubeEdge Authors. KubeEdge: Kubernetes Native Edge Computing Framework [Електронний ресурс]. – Cloud Native Computing Foundation, 2023. – Режим доступу: https://kubeedge.io/docs/. – Дата звернення: 01.04.2026.

32. EdgeX Foundry. EdgeX Foundry Documentation [Електронний ресурс]. – LF Edge Foundation, 2023. – Режим доступу: https://docs.edgexfoundry.org/. – Дата звернення: 01.04.2026.

33. Yi S., Li C., Li Q. A Survey of Fog Computing: Concepts, Applications and Issues // Proceedings of the 2015 Workshop on Mobile Big Data. – New York : ACM, 2015. – P. 37–42.