

КВАЛІФІКАЦІЙНА РОБОТА

Комп'ютерна система периферійних обчислень (edge computing)
для попередньої обробки потокових даних

Назва теми

Рівень вищої освіти перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Шифр, назва

Спеціальність 123 «Комп'ютерна інженерія»

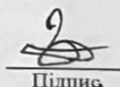
Шифр, назва

Освітня програма «Комп'ютерна інженерія та програмування»

Назва

Шифр КвРКІ 2304276.22.04.85 ПЗ

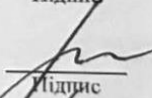
Виконав здобувач IV курсу, група KI2-22-4


Підпис

Данило ЗВАРИЧ

Ініціали, прізвище

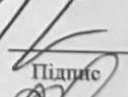
Керівник доктор техн. наук, професор
Науковий ступінь, учене звання


Підпис

Єлизавета ГНАТЧУК

Ініціали, прізвище

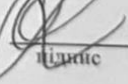
Нормоконтролер канд. фіз.-мат. наук, доц.
Науковий ступінь, учене звання


Підпис

Тетяна КИСЛІЬ

Ініціали, прізвище

До захисту допускаю:
завідувач кафедри КПС


Підпис

Ольга ПАВЛОВА

Ініціали, прізвище

«19» червня 2026 р.

дата

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Рівень вищої освіти ПЕРШИЙ (БАКАЛАВРСЬКИЙ)

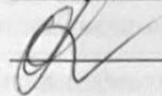
Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ

Завідувачка кафедри КІІС



Ольга ПАВЛОВА

“ 10 ” 01 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Зваричу Данилу Віталійовичу

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Комп'ютерна система периферійних обчислень (edge computing) для попередньої обробки поточкових даних

Керівник проекту (роботи) Гнатчук Єлизавета Геннадіївна, д.т.н., проф.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 7

2. Термін подання здобувачем роботи на кафедру 01.06.2026 р.

3. Вихідні дані до роботи Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) _____

Дослідження предметної області периферійних обчислень та формування задачі попередньої обробки поточкових даних

Проектування та моделювання програмно-апаратної системи периферійних обчислень для попередньої обробки поточкових даних

Програмно-апаратна реалізація системи периферійних обчислень для попередньої обробки поточкових даних

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) _____

Результати роботи програмного застосунку

Результати роботи апаратного edge-вузла на базі ESP32

Структурні схеми програмно-апаратної системи

6. Консультанти розділів кваліфікаційної роботи

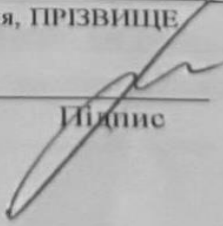
Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання « 10 » 01 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напрямку дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2026	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2026	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2026	виконано
4	Робота над розділом 2 – проектування та моделювання програмно-апаратної системи периферійних обчислень для попередньої обробки потокових даних	01.04.2026	виконано
5	Робота над розділом 3 – програмно-апаратна реалізація системи периферійних обчислень для попередньої обробки потокових даних	29.04.2026	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2026	виконано
7	Попередній захист ВКР	26.05.2026	виконано
8	Захист ВКР на засіданні ЕК	Червень 2026 року	

Здобувач  Підпис Данило ЗВАРИЧ
Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи  Підпис Єлизавета ГНАТЧУК
Ім'я, ПРІЗВИЩЕ

[illegible]

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Комп'ютерна система периферійних обчислень (edge computing) для попередньої обробки поточкових даних».

Автор роботи: Данило ЗВАРИЧ.

Керівник роботи: Єлизавета ГНАТЧУК.

Пояснювальна записка: 74 с., 12 рис., 1 табл., 63 джерела.

Графічна частина: 3 креслення.

ARDUINO, EDGE COMPUTING, ESP32, ІНТЕРНЕТ РЕЧЕЙ, МІКРОКОНТРОЛЕР, ПЕРИФЕРІЙНІ ОБЧИСЛЕННЯ, ПОТОКОВІ ДАНІ, ПРОГРАМНО-АПАРАТНА СИСТЕМА, РОЗПОДІЛЕНІ СИСТЕМИ, СИМУЛЯЦІЯ.

Кваліфікаційна робота бакалавра присвячена розробці та дослідженню системи попередньої обробки поточкових даних із використанням технологій периферійних обчислень. Актуальність теми зумовлена стрімким зростанням обсягів даних у системах Інтернету речей та необхідністю зменшення навантаження на централізовані обчислювальні ресурси.

Метою роботи є розробка програмно-апаратної системи, яка забезпечує імітацію надходження поточкових даних, їх попередню обробку на периферійному рівні та аналіз ефективності edge computing у порівнянні з традиційною централізованою обробкою. У роботі проведено аналіз концепції периферійних обчислень, досліджено особливості поточкових даних та розглянуто сучасні підходи до їх обробки.

Результатом роботи є програмно-апаратний прототип системи, що реалізує механізми локальної обробки поточкових даних на базі ESP32 та програмного застосунку C#. Розроблена система дозволяє досліджувати вплив периферійних обчислень на затримки, навантаження мережі та ефективність функціонування системи.



Підпис здобувача

30.05.2026

Дата

ЗМІСТ

Вступ.....	4
1 Дослідження предметної області та формування задачі	8
1.1 Концепція периферійних обчислень (edge computing) та їх місце в сучасних ІТ-системах.....	8
1.2 Особливості поточкових даних та вимоги до їх обробки в реальному часі	10
1.3 Архітектурні підходи до побудови систем попередньої обробки даних	13
1.4 Огляд сучасних технологій та інструментів обробки потоків.....	16
1.5 Дослідження існуючих рішень і їх обмежень	19
1.6 Формування мети та задач дослідження.....	22
1.7 Висновки до першого розділу	24
2 Проєктування та моделювання програмно-апаратної системи периферійних обчислень для попередньої обробки поточкових даних	26
2.1 Загальна концепція побудови системи попередньої обробки	26
2.2 Проєктування архітектури програмно-апаратної edge-системи	29
2.3 Моделювання потоків даних та сценаріїв їх надходження.....	32
2.4 Проєктування механізмів фільтрації, агрегації та трансформації даних	35
2.5 Вибір технологічного стеку.....	38
2.6 Опис взаємодії компонентів та обміну повідомленнями	41
2.7 Висновки до другого розділу	43
3 Програмно-апаратна реалізація системи периферійних обчислень для попередньої обробки поточкових даних.....	45
3.1 Реалізація модулів збору та прийому поточкових даних.....	45
3.2 Реалізація підсистеми попередньої обробки на edge-рівні	49
3.3 Реалізація механізмів фільтрації, агрегації та виявлення аномалій.....	52

					КвРКІ. 2304276.22.04.85 ПЗ			
Зм.	Арк.	№ док.ум.	Підпис	Дата	Комп'ютерна система периферійних обчислень (edge computing) для попередньої обробки поточкових даних. Пояснювальна записка	Літера	Аркуш	Аркушів
Виконав		Данило ЗВАРИЧ				у	2	72
Перевір.		Єлизавета ГНАТЧУК						
Н.контр.		Тетяна КИСІЛЬ				ХНУ КІ2-22-4		
Затвер.		Ольга ПАВЛОВА						

3.4 Інтеграція компонентів у єдину систему	55
3.5 Проведення експериментів із потоковими даними	57
3.6 Дослідження продуктивності та затримок обробки	60
3.7 Оцінка ефективності запропонованого рішення	64
3.8 Висновки до третього розділу	65
Висновки.....	67
Перелік джерел посилань	69
Додаток А Результати роботи програмного застосунку	75
Додаток Б Результати роботи апаратного edge-вузла на базі ESP32	76
Додаток В Структурні схеми програмно-апаратної системи	77

ВСТУП

У сучасних умовах розвитку інформаційних технологій спостерігається стрімке зростання обсягів даних, що генеруються різноманітними цифровими пристроями. Особливо це характерно для систем Інтернету речей, де численні сенсори, контролери та вбудовані системи безперервно формують потоки телеметричної інформації. Такі дані мають високу інтенсивність, нерівномірну структуру та потребують оперативної обробки, що ускладнює їх ефективне використання в традиційних централізованих системах.

Класичні підходи до обробки даних передбачають передачу всієї інформації до централізованих обчислювальних центрів або хмарних платформ. Проте зі збільшенням кількості джерел даних та швидкості їх генерації цей підхід поступово втрачає ефективність. Зокрема, виникають проблеми перевантаження мережеских каналів, збільшення затримок під час передачі, а також зростання обчислювального навантаження на серверну інфраструктуру. У результаті це може призводити до зниження швидкості реагування системи та зменшення загальної продуктивності.

У відповідь на зазначені виклики формується новий підхід до організації обчислень, а саме периферійні обчислення (edge computing). Його основна ідея полягає у перенесенні частини обчислювальних процесів ближче до джерела даних, тобто безпосередньо на периферію мережі. Це дозволяє виконувати первинну обробку, фільтрацію та аналіз інформації ще до її передачі в центральну систему. Такий підхід дає змогу зменшити обсяг переданих даних, скоротити затримки та більш ефективно використовувати доступні ресурси.

Особливо актуальним є застосування периферійних обчислень для обробки поточкових даних у режимі реального часу. Поточкові дані відрізняються безперервністю надходження та потребою швидкого реагування, що накладає додаткові вимоги до систем обробки. У таких умовах важливим є забезпечення

стабільної роботи системи навіть при значних навантаженнях, а також здатність виконувати обробку з мінімальними затримками.

Разом із тим, практична реалізація подібних підходів потребує не лише теоретичного обґрунтування, але й створення моделей, які дозволяють дослідити поведінку системи в різних умовах. Зокрема, доцільним є моделювання потоків даних та порівняння різних сценаріїв їх обробки, що дає можливість оцінити ефективність використання edge computing у конкретних ситуаціях.

Актуальність теми полягає у необхідності підвищення ефективності обробки поточкових даних в умовах зростаючого навантаження на програмно-апаратні системи, а також у потребі дослідження сучасних підходів до розподілених обчислень, зокрема периферійних обчислень. Крім того, актуальність зумовлена практичною необхідністю створення наочних програмно-апаратних засобів, які дозволяють продемонструвати переваги використання edge підходів у реальних сценаріях.

Метою даної роботи є розробка програмно-апаратної системи периферійних обчислень, яка забезпечує імітацію надходження поточкових даних, їх попередню обробку на периферійному рівні та дозволяє оцінити ефективність такого підходу в порівнянні з традиційною централізованою обробкою.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- проаналізувати особливості поточкових даних та підходи до їх попередньої обробки із використанням периферійних обчислень;
- визначити структуру програмно-апаратної системи попередньої обробки поточкових даних;
- розробити апаратний edge-вузол на базі ESP32 для імітації надходження поточкових даних та їх локальної обробки;
- реалізувати програмний застосунок для моделювання, візуалізації та порівняння режимів обробки даних;

- провести перевірку роботи системи у режимах без edge-обробки та з edge-обробкою;
- оцінити ефективність запропонованого рішення за показниками обсягу переданих даних, затримки та стабільності роботи.

Об'єктом дослідження є процеси обробки поточкових даних у розподілених інформаційних системах.

Предметом дослідження є методи, апаратні та програмні засоби попередньої обробки поточкових даних із використанням технологій периферійних обчислень.

Методи дослідження, використані у роботі, включають:

- аналіз наукових та технічних джерел для вивчення сучасних підходів;
- методи моделювання для побудови структури системи;
- методи програмної інженерії для реалізації програмного прототипу;
- методи програмно-апаратної реалізації для створення апаратного edge-вузла;
- експериментальні методи для оцінки ефективності роботи системи;
- порівняльний аналіз результатів роботи в різних режимах.

Наукова новизна одержаних результатів полягає у розробці моделі програмно-апаратної системи попередньої обробки поточкових даних, яка поєднує апаратний edge-вузол для локальної обробки даних та програмний застосунок для моделювання, візуалізації й порівняння режимів роботи. Запропонований підхід дозволяє наочно оцінити вплив периферійних обчислень на характеристики системи, зокрема обсяг переданих даних, затримки та стабільність обробки.

Практичне значення отриманих результатів полягає у створенні програмно-апаратного прототипу, який може бути використаний як демонстраційна модель для дослідження ефективності edge computing. Апаратний вузол на базі ESP32 забезпечує імітацію надходження поточкових

даних та їх локальну обробку, а програмний застосунок C# дозволяє моделювати, візуалізувати й аналізувати різні режими роботи системи.

Очікувані результати роботи включають:

- створення апаратного edge-вузла для імітації надходження та попередньої обробки потокових даних;
- реалізацію програмного застосунку для моделювання, візуалізації та аналізу потоків даних;
- отримання кількісних показників ефективності обробки;
- підтвердження доцільності використання edge computing для зменшення навантаження на систему;
- формування висновків щодо переваг та обмежень запропонованого підходу.

Структура роботи складається з трьох розділів. У першому розділі розглянуто теоретичні аспекти предметної області. Другий розділ присвячено проектуванню та моделюванню системи. У третьому розділі описано програмно-апаратну реалізацію та результати експериментальних досліджень. У висновках наведено основні результати роботи.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУВАННЯ ЗАДАЧІ

1.1 Концепція периферійних обчислень (edge computing) та їх місце в сучасних ІТ-системах

Упродовж останніх років характер роботи з даними помітно змінився. Якщо раніше основні обсяги інформації формувалися у відносно контрольованих системах і передавалися у централізовані середовища для подальшої обробки, то сьогодні ситуація виглядає інакше. Дані генеруються повсюдно: мобільні пристрої, датчики, вбудовані системи, транспортні засоби постійно створюють потоки інформації, які не завжди доцільно обробляти централізовано [1, 2].

Традиційна модель, що базується на передачі всіх даних у хмару, довгий час залишалася універсальним рішенням. Вона добре масштабувалася, дозволяла централізувати управління та спрощувала обчислювальні процеси. Але разом із перевагами з часом стали проявлятися і слабкі місця. Збільшення кількості джерел даних призвело до того, що канали передачі почали перевантажуватися, а затримки будуть зростати. У деяких сценаріях це перестало бути просто незручністю і стало реальним обмеженням для системи [3].

Особливо це відчутно у випадках, коли рішення потрібно приймати швидко. Наприклад, якщо мова йде про обробку сигналів із сенсорів у промисловому середовищі або про системи моніторингу, де важлива реакція в режимі, близькому до реального часу. Передача даних у хмару, їх обробка і повернення результату можуть займати більше часу, ніж допустимо для таких задач [4].

Саме в цьому контексті почали активно розвиватися периферійні обчислення. Ідея виглядає доволі простою: частину обробки переносимо ближче до місця, де виникають дані. Тобто замість того, щоб одразу відправляти весь

потік у центр, система спочатку щось робить із ним локально. Це може бути відсікання шуму, агрегація або навіть базовий аналіз [5].

Важливий момент полягає в тому, що edge computing не варто розглядати як заміну хмарних обчислень. Швидше це додатковий рівень, який з'являється між джерелом даних і центральною системою. У результаті формується більш гнучка структура, де кожен рівень виконує свою роль. Периферія бере на себе швидкі та прості операції, а хмара виконує більш складні задачі, які потребують значних ресурсів [6].

Однією з причин популярності такого підходу є можливість зменшити затримки. Коли обробка виконується ближче до джерела, система швидше реагує на події. У багатьох випадках це дозволяє уникнути ситуацій, коли рішення запізнюється і вже не має практичної цінності. Це особливо актуально для задач керування або моніторингу [7].

Окрім цього, важливим є зменшення обсягу даних, які передаються мережею. Якщо на периферійному рівні вже відбувається фільтрація або узагальнення інформації, то до хмари потрапляє не повний потік, а лише його частина. Це дозволяє знизити навантаження на мережу і водночас зробити подальшу обробку більш осмисленою, оскільки система працює вже з підготовленими даними [8].

Разом із тим, не можна сказати, що edge computing є універсальним рішенням без недоліків. Перенесення обробки на периферію означає, що частина логіки виконується на пристроях із обмеженими ресурсами. Це накладає обмеження на складність алгоритмів і вимагає більш обережного підходу до проектування. Крім того, ускладнюється загальна структура системи, адже з'являється більше компонентів, які потрібно узгоджувати між собою [9].

Ще один аспект, який варто враховувати, є різноманітність середовищ, у яких працюють периферійні вузли. На відміну від хмарних серверів, де умови більш-менш стандартизовані, edge пристрої можуть значно відрізнятися за

характеристиками. Це створює додаткові труднощі при розробці універсальних рішень, які повинні працювати стабільно в різних умовах [10].

Попри зазначені складності, інтерес до периферійних обчислень продовжує зростати. Це пов'язано з тим, що сучасні ІТ-системи дедалі частіше працюють із потоковими даними, а також потребують швидкої реакції на події. У таких умовах комбінування edge та хмарних підходів виглядає логічним і практично виправданим.

Якщо розглядати місце edge computing у сучасній архітектурі, то його можна уявити як проміжний рівень, який виконує функцію попередньої обробки. Він не замінює інші компоненти системи, але змінює характер їхньої взаємодії. Дані перестають бути просто «сировиною», що передається в центр, і починають оброблятися ще до того, як потрапляють у хмару.

У результаті змінюється і підхід до проектування систем. Замість жорстко централізованої моделі формується більш розподілена структура, у якій важливу роль відіграє локальна обробка. Це дозволяє системам бути більш адаптивними, краще реагувати на зміну умов і ефективніше використовувати ресурси.

Таким чином, периферійні обчислення займають важливе місце в сучасних ІТ-системах. Вони не є окремою альтернативою існуючим підходам, а скоріше доповнюють їх, розширюючи можливості обробки даних. Їх використання особливо доцільне у тих випадках, коли важливими є швидкість, обсяг переданих даних та здатність системи працювати в умовах обмежених ресурсів.

1.2 Особливості поточкових даних та вимоги до їх обробки в реальному часі

Потокові дані є однією з ключових характеристик сучасних інформаційних систем. На відміну від традиційних пакетних підходів, де дані накопичуються і обробляються періодично, потокові дані надходять безперервно і часто потребують негайної реакції. Це змінює сам підхід до їх обробки, оскільки система

					КвРКІ.2304276.22.04.85 ПЗ	Арк.
						10
Зм.	Арк.	№ докум.	Підпис	Дата		

повинна працювати не з завершеними наборами, а з постійно оновлюваним потоком інформації [11].

Однією з основних особливостей потокових даних є їхня потенційно необмежена довжина. Потік не має чітко визначеного початку або кінця, що ускладнює використання класичних алгоритмів обробки. У таких умовах замість повного аналізу всього набору даних застосовуються підходи, орієнтовані на обробку частин потоку, наприклад, через ковзні вікна або мікропакети [12]. Це дозволяє отримувати результати без необхідності зберігати всі попередні значення.

Ще однією важливою характеристикою є висока швидкість надходження даних. У реальних системах кількість подій може вимірюватися сотнями або тисячами на секунду. При цьому навантаження може змінюватися нерівномірно, що створює додаткові труднощі для систем обробки. У таких умовах важливо забезпечити стабільність роботи навіть при пікових навантаженнях [13].

Також слід враховувати, що поточкові дані часто містять шум або некоректні значення. Наприклад, сенсори можуть передавати помилкові показники через фізичні обмеження або збої. Це означає, що система повинна не лише обробляти дані, але й виконувати їх попередню очистку. У багатьох випадках фільтрація стає необхідним етапом перед будь-якою аналітикою [14].

Окремо варто згадати проблему затримок. У потокових системах важливо не тільки обробити дані, але зробити це достатньо швидко. Існує поняття «латентності», яке описує час від моменту отримання події до отримання результату її обробки. У задачах реального часу цей показник має критичне значення. Наприклад, у системах моніторингу затримка може визначати, наскільки швидко система реагує на зміну стану [15].

Ще одна особливість пов'язана з порядком надходження даних. У реальних мережах події можуть приходити не у тому порядку, в якому вони були згенеровані. Це ускладнює обробку, оскільки система повинна враховувати можливість перестановки подій або навіть їх втрати. У таких випадках

використовуються механізми буферизації або маркування часу, які дозволяють частково вирішити цю проблему [16].

Важливо також враховувати обмеження ресурсів. Потоківі системи часто працюють у середовищах, де доступна пам'ять або обчислювальна потужність є обмеженими. Це означає, що алгоритми повинні бути достатньо легкими та ефективними. Надмірно складні підходи можуть просто не працювати у реальних умовах, особливо якщо мова йде про периферійні пристрої [17].

У зв'язку з цим формуються певні вимоги до систем обробки поточкових даних. По-перше, це здатність працювати у режимі реального часу або близькому до нього. Система повинна обробляти події швидше, ніж вони надходять, інакше виникає накопичення черги та втрата актуальності результатів [18]. По-друге, важливою є масштабованість, оскільки обсяг даних може зростати без попередження.

Ще однією вимогою є стійкість до помилок. Потоківі системи повинні продовжувати роботу навіть у випадку часткових збоїв. Це може досягатися за рахунок резервування, повторної обробки або використання розподілених підходів. Водночас такі механізми не повинні значно збільшувати затримку [19].

Крім того, важливим є баланс між точністю та швидкістю. У багатьох випадках система може працювати з наближеними значеннями, якщо це дозволяє значно зменшити час обробки. Наприклад, використання агрегованих даних або усереднення може бути достатнім для прийняття рішень у реальному часі [20].

У контексті периферійних обчислень ці вимоги набувають додаткового значення. Оскільки обробка переноситься на edge рівень, система повинна працювати ще більш ефективно, використовуючи обмежені ресурси. Це означає, що багато рішень повинні бути максимально простими, але водночас достатньо інформативними.

Якщо розглядати це з практичної точки зору, то саме такі вимоги визначають структуру сучасних систем обробки потоків. Вони впливають на

вибір алгоритмів, архітектури та технологій. У результаті формується підхід, у якому обробка відбувається поступово, з урахуванням як швидкості надходження даних, так і доступних ресурсів.

Потокові дані мають ряд особливостей, які суттєво відрізняють їх від традиційних наборів даних. Вони потребують спеціалізованих підходів до обробки, орієнтованих на швидкість, гнучкість та ефективність. Саме ці характеристики визначають вимоги до сучасних систем, зокрема тих, що використовують периферійні обчислення.

Класифікація основних типів поточкових даних представлена на рисунку 1.1.



Рисунок 1.1 – Класифікація поточкових даних

Як видно зі схеми, поточкові дані можуть відрізнятися як за джерелом формування, так і за характером надходження та обробки. Це безпосередньо впливає на вибір архітектурних підходів і механізмів їх попередньої обробки.

1.3 Архітектурні підходи до побудови систем попередньої обробки даних

Побудова систем попередньої обробки даних тісно пов'язана з вибором архітектурного підходу. Саме архітектура визначає, яким чином дані рухаються всередині системи, де виконуються обчислення та як взаємодіють окремі

компоненти. У випадку потокових даних ці питання набувають особливого значення, оскільки обробка повинна відбуватися безперервно і з мінімальними затримками [21].

Традиційно обробка даних будувалася за централізованою моделлю, де всі обчислення виконуються на серверній стороні. У такій архітектурі клієнтські пристрої виконують роль джерел даних, а основна логіка зосереджена у центрі. Подібний підхід є відносно простим у реалізації та управлінні, однак він погано масштабується при значному зростанні кількості джерел даних [22]. Крім того, він створює залежність від стабільності мережі та пропускної здатності каналів.

З розвитком розподілених систем з'явилися підходи, які передбачають розподіл обчислювального навантаження між кількома вузлами. Одним із таких підходів є використання багаторівневих архітектур, де обробка даних виконується на різних рівнях – від джерела до центрального сервера. У таких системах дані можуть проходити кілька етапів трансформації, перш ніж потрапити до кінцевого сховища [23].

Окреме місце займають архітектури, орієнтовані на події (event driven). У них взаємодія між компонентами відбувається через передачу повідомлень або подій. Кожна подія може ініціювати виконання певної логіки, що дозволяє будувати системи, які добре реагують на зміни у потоці даних. Такий підхід часто використовується у системах обробки потоків, оскільки він природно відповідає їхній природі [24].

Ще одним важливим напрямом є мікросервісна архітектура. Вона передбачає розділення системи на невеликі незалежні компоненти, кожен з яких виконує окрему функцію. Це дозволяє гнучко масштабувати окремі частини системи та спрощує її розвиток. У контексті обробки даних мікросервіси можуть відповідати за різні етапи обробки, такі як прийом даних, фільтрація, агрегація або збереження [25].

У системах потокової обробки часто застосовуються спеціалізовані архітектурні моделі, наприклад, Lambda та Kappa архітектури. Lambda-

архітектура поєднує пакетну та потокову обробку, дозволяючи отримувати як точні, так і швидкі результати. Водночас вона є досить складною в реалізації через необхідність підтримки двох різних шляхів обробки даних [26].

Архітектура Карра, у свою чергу, пропонує спрощений підхід, у якому вся обробка виконується у вигляді потоку. Це дозволяє уникнути дублювання логіки, але вимагає більш продуманих механізмів обробки та збереження даних [27]. Вибір між цими підходами залежить від конкретних вимог до системи та характеру даних.

У контексті периферійних обчислень особливу роль відіграють гібридні архітектури, які поєднують edge та cloud рівні. У таких системах частина обробки виконується локально, а частина – у централізованих середовищах. Це дозволяє досягти балансу між швидкістю обробки та доступністю обчислювальних ресурсів [28].

При проектуванні систем попередньої обробки важливим є також вибір способу передачі даних між компонентами. Це може бути синхронна взаємодія або асинхронна передача через черги повідомлень. У потокових системах частіше використовується асинхронний підхід, оскільки він дозволяє уникнути блокувань і забезпечує кращу масштабованість [29].

Не менш важливим є питання масштабування. Система повинна мати можливість обробляти зростаючий обсяг даних без суттєвого зниження продуктивності. Для цього використовуються механізми горизонтального масштабування, коли нові вузли додаються до системи за потреби. Це особливо актуально для розподілених архітектур [30].

Ще одним аспектом є забезпечення відмовостійкості. У розподілених системах окремі компоненти можуть виходити з ладу, тому архітектура повинна передбачати механізми відновлення або дублювання функцій. Це дозволяє системі продовжувати роботу навіть у випадку часткових збоїв [31].

У випадку попередньої обробки даних важливу роль відіграє також локалізація обчислень. Чим ближче до джерела даних виконується обробка, тим

менше навантаження створюється на мережу. Це є одним із ключових принципів edge computing і впливає на вибір архітектурного підходу [32].

Якщо узагальнити, сучасні системи попередньої обробки даних рідко використовують лише один архітектурний підхід. Зазвичай це комбінація кількох моделей, адаптованих під конкретні умови. Наприклад, система може поєднувати event driven взаємодію з мікросервісною структурою та використовувати edge рівень для первинної обробки.

У практичному плані це означає, що при проектуванні системи необхідно враховувати не лише функціональні вимоги, але й обмеження середовища, у якому вона працює. Саме архітектура визначає, наскільки ефективно система зможе працювати з потоковими даними та чи зможе вона адаптуватися до змін навантаження.

Таким чином, вибір архітектурного підходу є одним із ключових етапів побудови систем попередньої обробки даних. Він впливає на продуктивність, масштабованість та надійність системи, а також визначає можливості її подальшого розвитку.

1.4 Огляд сучасних технологій та інструментів обробки потоків

Розвиток поточкових даних безпосередньо вплинув на появу великої кількості спеціалізованих технологій, орієнтованих на їх обробку. Якщо раніше системи будувалися навколо пакетної обробки, то сьогодні дедалі частіше використовуються інструменти, які здатні працювати з безперервними потоками інформації. Це зумовлено як зростанням обсягів даних, так і вимогами до швидкості отримання результатів.

Одним із ключових елементів сучасних систем потокової обробки є платформи передачі повідомлень. Вони забезпечують надійне та масштабоване транспортування даних між компонентами. Найбільш відомим рішенням у цій категорії є Apache Kafka, який використовується як розподілена система обміну

повідомленнями та дозволяє обробляти великі обсяги даних у режимі реального часу [33, 34]. Kafka фактично виступає центральною «шиною даних», через яку проходять потоки подій.

Окрім Kafka, існують і інші рішення для організації обміну повідомленнями, наприклад RabbitMQ або Apache Pulsar. Вони також підтримують асинхронну взаємодію між компонентами та дозволяють будувати системи, які добре масштабуються. Вибір конкретного інструменту часто залежить від вимог до продуктивності, затримок та моделі доставки повідомлень [35, 36].

Наступною важливою групою технологій є системи обробки потоків. Вони виконують безпосередню обробку даних, що надходять у вигляді потоку. Одним із найпоширеніших інструментів є Apache Flink, який підтримує як потокову, так і пакетну обробку та дозволяє працювати з подіями у реальному часі [37, 38]. Його особливістю є можливість обробки подій з урахуванням часу їх виникнення, що важливо для коректного аналізу.

Ще одним популярним рішенням є Apache Spark Streaming, який є розширенням платформи Spark і дозволяє виконувати обробку поточкових даних у вигляді мікропакетів. Такий підхід спрощує реалізацію, але може призводити до збільшення затримок у порівнянні з повністю поточковими системами [39, 40]. Вибір між Flink і Spark часто залежить від вимог до латентності та складності обробки.

Окремо варто згадати Apache Storm, який був одним із перших інструментів для потокової обробки. Хоча його популярність дещо знизилася, він все ще використовується у певних сценаріях, де потрібна проста та швидка обробка подій [41].

Крім систем обробки, важливу роль відіграють інструменти для побудови поточкових конвеєрів (pipelines). Наприклад, Apache Beam пропонує універсальну модель програмування, яка може виконуватися на різних платформах, включаючи Flink та Spark [42, 43]. Це дозволяє розробникам створювати переносимі рішення без жорсткої прив'язки до конкретної технології.

У контексті периферійних обчислень дедалі більшої популярності набувають легковагові рішення, які можуть працювати безпосередньо на edge пристроях. До них можна віднести різні stream processing бібліотеки, а також вбудовані платформи для обробки даних. У таких випадках акцент робиться не на максимальній продуктивності, а на ефективності використання ресурсів [44].

Ще одним важливим напрямом є використання контейнеризації та оркестрації. Технології, такі як Docker і Kubernetes, дозволяють розгортати та масштабувати компоненти систем обробки потоків у розподіленому середовищі [45, 46]. Це значно спрощує управління складними системами та забезпечує їхню гнучкість.

Для роботи з поточковими даними також широко використовуються системи зберігання, оптимізовані під швидкий запис і читання. Наприклад, бази даних типу time series дозволяють ефективно працювати з часовими рядами, що часто зустрічаються у поточкових сценаріях [47]. Вони забезпечують швидкий доступ до даних і підтримують агрегаційні операції.

Важливою частиною сучасних систем є також інструменти моніторингу та візуалізації. Вони дозволяють відстежувати стан системи, аналізувати потоки даних і швидко реагувати на зміни. До таких інструментів можна віднести Grafana або Prometheus, які широко використовуються для аналізу метрик [48].

Окремо варто звернути увагу на підходи до обробки даних безпосередньо на пристроях. У цьому випадку використовуються спрощені алгоритми, які дозволяють виконувати базову обробку без значного навантаження. Це може бути фільтрація, агрегація або простий аналіз аномалій. Подібні рішення добре вписуються у концепцію edge computing [49].

У реальних системах зазвичай використовується комбінація кількох технологій. Наприклад, дані можуть передаватися через Kafka, оброблятися у Flink, а результати зберігатися у спеціалізованих базах даних. Такий підхід дозволяє використовувати сильні сторони кожного інструменту та будувати гнучкі системи [50].

Не існує універсального рішення, яке підходить для всіх випадків. Вибір технологій залежить від конкретних вимог до системи: швидкості обробки, обсягу даних, доступних ресурсів та складності логіки. У випадку edge систем ці вимоги стають ще більш жорсткими, оскільки доводиться працювати в умовах обмежених ресурсів.

Таким чином, сучасні технології обробки поточкових даних пропонують широкий набір інструментів для побудови ефективних систем. Вони дозволяють організувати обробку як на централізованому, так і на периферійному рівні, що відкриває можливості для створення гнучких і масштабованих рішень.

1.5 Дослідження існуючих рішень і їх обмежень

На сьогодні існує велика кількість рішень, орієнтованих на обробку поточкових даних. Вони відрізняються за підходами, архітектурою та областями застосування. Частина з них розроблена для роботи у хмарних середовищах, тоді як інші призначені для локальної або гібридної обробки. Проте попри різноманіття інструментів, жодне з рішень не можна назвати універсальним.

Багато сучасних систем обробки потоків побудовані навколо централізованої або квазіцентралізованої моделі. Наприклад, такі платформи, як Apache Spark Streaming або Apache Flink, дозволяють ефективно обробляти великі обсяги даних у реальному часі [51]. Вони забезпечують масштабованість, підтримують складні операції над потоками та добре інтегруються з іншими компонентами екосистеми. Водночас їх використання часто передбачає наявність достатньо потужної інфраструктури, що не завжди є можливим у випадках, коли обробка повинна виконуватися ближче до джерела даних.

Крім того, навіть у поточкових системах, які позиціонуються як real time, затримка обробки не завжди є мінімальною. Наприклад, у випадку мікропакетної обробки, як у Spark Streaming, події обробляються невеликими групами, що може призводити до накопичення затримки. У багатьох сценаріях це є допустимим,

але для задач, де важлива миттєва реакція, подібний підхід виглядає не найкращим [52].

Існують також рішення, орієнтовані на передачу даних, такі як Apache Kafka або RabbitMQ. Вони добре справляються із задачею доставки повідомлень і можуть працювати з великими потоками подій [53]. Проте ці системи не виконують повноцінну обробку даних, а лише забезпечують інфраструктуру для їх передачі. Це означає, що додаткові компоненти для обробки все одно необхідні, що ускладнює загальну архітектуру.

У контексті периферійних обчислень пропонуються різні підходи до розміщення логіки на edge рівні. Наприклад, концепції fog computing або multi access edge computing передбачають використання проміжних вузлів для виконання частини обчислень [54]. Такі рішення дозволяють зменшити затримки та навантаження на мережу, однак вони часто потребують спеціалізованої інфраструктури та складного налаштування.

Ще одним обмеженням існуючих рішень є їхня складність. Багато сучасних платформ мають широкий функціонал, але водночас вимагають значних зусиль для розгортання та підтримки. Це стосується як налаштування середовища, так і розробки самої логіки обробки. У результаті такі системи можуть бути надлишковими для задач, де потрібна лише базова попередня обробка даних [55].

Також варто звернути увагу на проблему адаптації під обмежені ресурси. Більшість популярних інструментів розраховані на серверні середовища і не завжди ефективно працюють на периферійних пристроях. Це означає, що їх використання на edge рівні може бути обмеженим або вимагати суттєвого спрощення логіки [56].

Окремо можна виділити питання гнучкості. Деякі системи добре працюють у конкретних сценаріях, але важко адаптуються до змін вимог. Наприклад, зміна формату даних або логіки обробки може вимагати значних

змін у конфігурації або коді. Це ускладнює використання таких рішень у динамічних середовищах, де вимоги можуть змінюватися досить швидко.

Ще один момент, який не завжди очевидний, – це відсутність наочності. Багато промислових систем орієнтовані на продуктивність і масштабованість, але не на демонстрацію роботи. Це означає, що їх складно використовувати як навчальні або дослідницькі інструменти, оскільки внутрішні процеси залишаються «за кадром» [57].

Таблиця 1.1 – Порівняння існуючих аналогів

Рішення / підхід	Основні переваги	Основні недоліки	Особливості застосування
Apache Spark Streaming	Масштабованість, інтеграція з Big Data екосистемою	Мікропакетна обробка створює затримки	Підходить для великих централізованих систем
Apache Flink	Висока швидкість поточної обробки, підтримка real time	Високі вимоги до інфраструктури	Ефективний для складної аналітики потоків
Apache Kafka	Надійна передача великих потоків повідомлень	Не виконує повноцінну обробку даних	Використовується як транспортний рівень
RabbitMQ	Простота організації черг повідомлень	Обмежені можливості поточної аналітики	Підходить для систем обміну повідомленнями
Fog Computing	Зменшення затримок, часткова локальна обробка	Складність розгортання інфраструктури	Використовується у розподілених IoT системах

Кінець таблиці 1.1

Multi access Edge Computing (MEC)	Обробка даних ближче до користувача	Потребує спеціалізованого обладнання	Актуально для телекомунікаційн их мереж
Централізо вана обробка	Простота централізованого контролю	Високе навантаження на мережу та сервери	Ефективна при стабільних каналах зв'язку
Edge підхід	Мінімізація затримок та трафіку	Обмежені ресурси периферійних пристроїв	Доцільний для систем реального часу

У підсумку можна сказати, що існуючі рішення добре вирішують окремі задачі, але не завжди підходять для побудови компактних і наочних систем попередньої обробки даних. Частина з них є занадто складною, інші – не враховують обмеження периферійних пристроїв, а деякі взагалі орієнтовані лише на передачу даних без їх обробки.

Саме ці обмеження створюють передумови для розробки більш простих і адаптованих моделей, які поєднують базову обробку даних із можливістю наочного аналізу результатів. У таких моделях важливо не лише забезпечити роботу алгоритмів, але й зробити систему зрозумілою для дослідження та демонстрації.

1.6 Формування мети та задач дослідження

Проведений аналіз предметної області показує, що сучасні засоби дедалі частіше працюють із потоковими даними, які надходять у великому обсязі та з високою швидкістю. При цьому традиційні підходи до обробки, що базуються на централізованих моделях, не завжди забезпечують необхідну ефективність.

Зокрема, виникають проблеми із затримками, перевантаженням мережі та обробкою надлишкової інформації.

Розглянуті архітектурні підходи та технології дозволяють частково вирішити ці проблеми, однак більшість існуючих аналогів орієнтовані або на великі розподілені системи, або на вузькі прикладні задачі. У багатьох випадках вони є складними для розгортання, потребують значних ресурсів або не враховують особливості периферійної обробки даних.

Окремо варто зазначити, що значна частина існуючих систем не орієнтована на наочну демонстрацію процесів обробки. Це ускладнює їх використання в навчальних або дослідницьких цілях, де важливо не лише отримати результат, але й зрозуміти, як саме він формується.

У результаті виникає потреба у створенні компактної моделі системи, яка дозволяє дослідити процес попередньої обробки поточкових даних у контексті периферійних обчислень. Така модель повинна бути достатньо простою для розуміння, але водночас відображати основні принципи роботи реальних систем.

З урахуванням зазначеного, метою даної роботи є розробка програмно-апаратної системи периферійних обчислень, яка забезпечує імітацію надходження поточкових даних, їх попередню обробку на периферійному рівні та дозволяє оцінити ефективність такого підходу у порівнянні з традиційною централізованою обробкою.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- проаналізувати особливості поточкових даних та підходи до їх попередньої обробки із використанням периферійних обчислень;
- визначити структуру програмно-апаратної системи попередньої обробки поточкових даних;
- розробити апаратний edge-вузол на базі ESP32 для імітації надходження поточкових даних та їх локальної обробки;
- реалізувати програмний застосунок для моделювання, візуалізації та порівняння режимів обробки даних;

- провести перевірку роботи системи в режимах без edge-обробки та з edge-обробкою;
- оцінити ефективність запропонованого рішення за показниками обсягу переданих даних, затримки та стабільності роботи.

Об’єктом дослідження є процеси обробки поточкових даних у сучасних інформаційних системах.

Предметом дослідження є методи та програмні засоби попередньої обробки поточкових даних із використанням периферійних обчислень.

Очікується, що розроблена система дозволить наочно продемонструвати відмінності між різними підходами до обробки даних, а також оцінити їх вплив на такі характеристики, як обсяг переданої інформації, затримка та загальна продуктивність.

Отримані результати можуть бути використані для кращого розуміння принципів побудови систем обробки поточкових даних, а також як основа для подальшого розвитку більш складних рішень у цій області.

1.7 Висновки до першого розділу

У першому розділі проведено аналіз предметної області попередньої обробки поточкових даних у середовищах периферійних обчислень. Розглянуто концепцію edge computing та визначено її роль у сучасних інформаційних системах.

Проаналізовано особливості поточкових даних і основні вимоги до систем їх обробки. Визначено, що такі дані характеризуються безперервністю надходження та потребують оперативної обробки в умовах обмежених ресурсів.

Також розглянуто основні архітектурні підходи та сучасні технології потокової обробки даних, зокрема Apache Kafka, Apache Flink, Spark Streaming і RabbitMQ. Аналіз існуючих рішень показав, що більшість із них орієнтовані на

серверні середовища та мають обмеження при використанні на периферійних пристроях.

У результаті визначено необхідність розробки компактної програмно-апаратної системи для попередньої обробки поточкових даних із використанням периферійних обчислень, а також сформульовано мету та задачі дослідження.

					КвРКІ.2304276.22.04.85 ПЗ	Арк.
						25
Зм.	Арк.	№ докум.	Підпис	Дата		

2 ПРОЄКТУВАННЯ ТА МОДЕЛЮВАННЯ ПРОГРАМНО-АПАРАТНОЇ СИСТЕМИ ПЕРИФЕРІЙНИХ ОБЧИСЛЕНЬ ДЛЯ ПОПЕРЕДНЬОЇ ОБРОБКИ ПОТОКОВИХ ДАНИХ

2.1 Загальна концепція побудови системи попередньої обробки

На етапі проєктування системи попередньої обробки поточкових даних важливо визначити загальну концепцію її побудови. Розроблювана система розглядається як програмно-апаратний прототип, що поєднує апаратний периферійний вузол для локальної обробки даних та програмний застосунок для моделювання, візуалізації й аналізу результатів роботи.

Основна ідея побудови системи полягає у перенесенні частини обробки ближче до джерела даних. Для цього використовується апаратний edge-вузол на базі ESP32, який імітує надходження поточкових даних, виконує їх первинну обробку та формує результати для подальшого аналізу. В межах апаратної частини потенціометр використовується як джерело змінного потоку даних, а кнопка забезпечує перемикання режимів роботи системи.

Програмна частина системи реалізується у вигляді застосунку мовою C#, який використовується для моделювання поточкових даних, відображення результатів обробки, порівняння режимів роботи та оцінки ефективності периферійних обчислень. Такий підхід дозволяє зберегти наочність дослідження і водночас показати практичну реалізацію edge-рівня на основі апаратного вузла.

У рамках запропонованої концепції система включає такі основні складові: джерело поточкових даних, апаратний edge-вузол, програмний модуль моделювання та аналізу, умовний центральний рівень прийому даних і підсистему збору метрик. Кожен із цих компонентів виконує окрему функцію в загальному процесі обробки.

Джерело даних відповідає за формування потоку подій. У реальних умовах це можуть бути фізичні сенсори або інші пристрої, що передають телеметричні дані. У межах даної роботи для імітації такого джерела використовується

апаратний edge-вузол на базі ESP32, у якому потенціометр формує змінне вхідне значення, а програмний застосунок додатково забезпечує моделювання та аналіз потоків даних. Це дозволяє досліджувати роботу системи як на рівні апаратного вузла, так і на рівні програмної моделі.

Наступним етапом є попередня обробка даних. Саме тут реалізується концепція периферійних обчислень. Дані не передаються одразу далі, а спочатку проходять через набір операцій, серед яких фільтрація, агрегація та виявлення аномалій. Такий підхід дозволяє зменшити обсяг інформації, що передається, а також підготувати її до подальшого аналізу.

Після обробки дані передаються до центрального компонента системи, який умовно представляє хмарну інфраструктуру. В межах даної роботи цей компонент не виконує складної обробки, а використовується для моделювання прийому даних і оцінки навантаження. Це дозволяє зосередитися саме на етапі попередньої обробки, не ускладнюючи систему зайвими деталями.

Важливою складовою є підсистема збору метрик. Вона відповідає за фіксацію ключових показників роботи системи, таких як кількість згенерованих подій, кількість переданих даних, затримка та пропускна здатність. Саме ці показники використовуються для оцінки ефективності обраного підходу.

При побудові загальної концепції також важливо врахувати зв'язок між апаратною та програмною частинами системи. Апаратний edge-вузол відповідає за формування вхідного потоку та виконання локальної обробки даних, а програмний застосунок використовується для відображення результатів і порівняння різних режимів роботи. Такий поділ дозволяє зберегти простоту системи та водночас показати взаємодію периферійного пристрою з програмним середовищем.

Для наочного представлення структури системи на рисунку 2.1 представлена схема, яка відображає взаємодію апаратної та програмної частин програмно-апаратної системи.

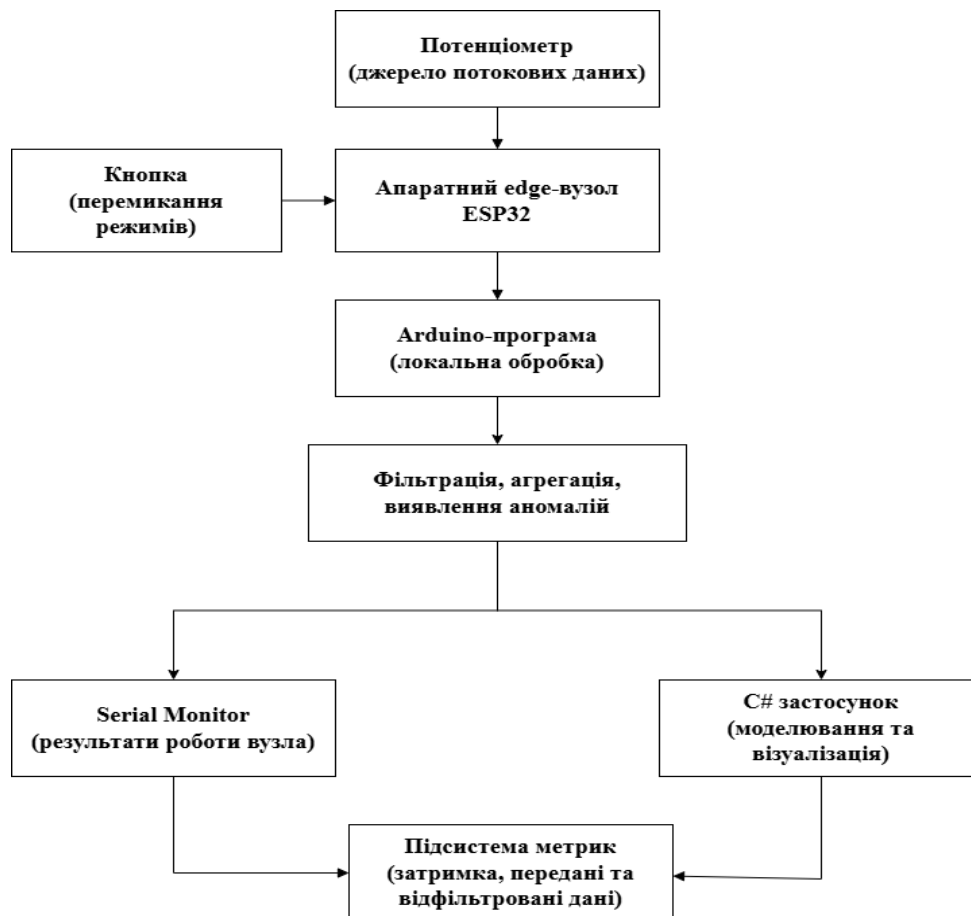


Рисунок 2.1 – Загальна структура програмно-апаратної системи попередньої обробки потокових даних

На рисунку умовно показано, як дані проходять через систему: від генерації до обробки та передачі. Окремо виділено апаратний edge-вузол на базі ESP32, який виконує локальну обробку потоку даних, а також програмний застосунок, що використовується для моделювання, візуалізації та оцінки результатів роботи системи.

Звертає на себе увагу те, що система може працювати у різних режимах. У першому випадку дані передаються без попередньої обробки, що відповідає класичному централізованому підходу. У другому випадку активується edge-рівень, і дані проходять локальну обробку перед подальшою передачею або аналізом. Такий поділ дозволяє безпосередньо порівнювати результати роботи системи в різних умовах.

Ще одним важливим моментом є організація потоку виконання. Система побудована таким чином, щоб обробка даних відбувалася поступово, без блокування основних процесів. Апаратний вузол виконує первинну обробку даних, а програмний застосунок забезпечує відображення результатів, збір метрик та аналіз ефективності роботи.

Загальна концепція також передбачає можливість зміни режиму роботи системи. Для цього в апаратній частині використовується кнопка, яка дозволяє перемикати режим без edge-обробки та режим із попередньою обробкою. Це дає змогу досліджувати поведінку системи у різних умовах і спостерігати, як змінюються її характеристики.

У підсумку запропонована концепція побудови системи поєднує простоту реалізації з достатнім рівнем деталізації. Вона дозволяє відобразити ключові принципи периферійних обчислень та забезпечує основу для подальшого проектування апаратної і програмної частин системи.

2.2 Проектування архітектури програмно-апаратної edge-системи

Після визначення загальної концепції побудови системи наступним кроком є уточнення її внутрішньої структури. На цьому етапі важливо не лише окреслити складові системи, але й зрозуміти, як саме вони взаємодіють між собою в процесі обробки даних. Фактично мова йде про перехід від загальної ідеї до більш конкретної організації роботи системи.

Якщо розглядати систему узагальнено, вона будується навколо послідовного проходження даних через кілька етапів. Спочатку формується потік подій, далі він проходить попередню обробку, після чого передається до центрального компонента. Паралельно з цим відбувається збір метрик, які дозволяють оцінити ефективність роботи системи. Така організація виглядає досить очевидною, однак саме вона дозволяє чітко відокремити відповідальність окремих частин системи.

Окремим елементом цієї архітектури є апаратний edge-вузол, який відповідає за отримання вхідних даних, виконання локальної обробки та формування результатів для подальшого аналізу. Структурну схему апаратного edge-вузла на базі ESP32 наведено на рисунку 2.2.

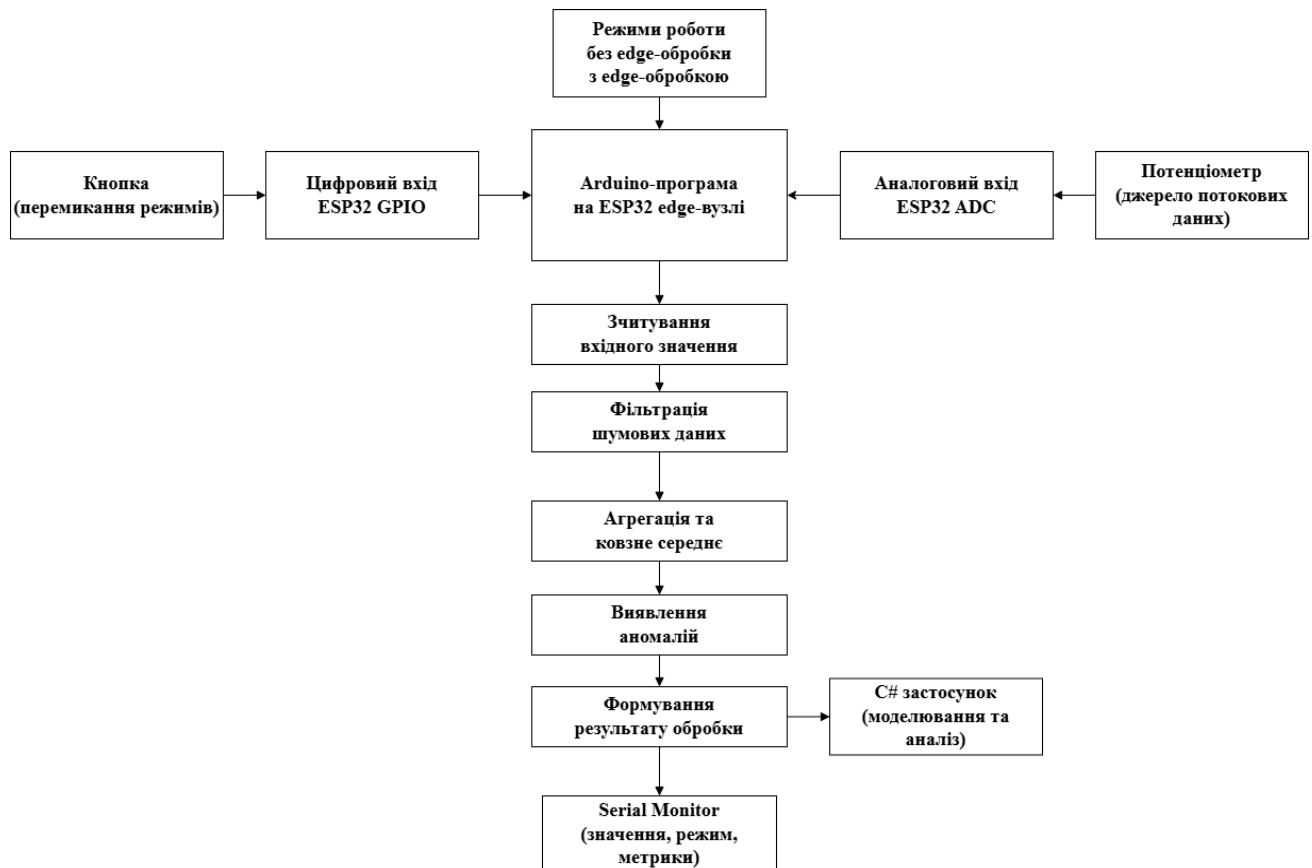


Рисунок 2.2 – Структурна схема апаратного edge-вузла на базі ESP32

На схемі показано основні компоненти апаратного edge-вузла та зв'язки між ними. Потенціометр використовується як джерело потокових даних, кнопка забезпечує перемикання режимів роботи, а ESP32 виконує локальну обробку вхідних значень за допомогою Arduino-програми.

Першим елементом архітектури є джерело потокових даних. У межах програмно-апаратної системи цю роль виконує потенціометр, значення якого зчитується апаратним edge-вузлом на базі ESP32. Такий підхід дозволяє наблизити модель до реальних умов, у яких дані надходять від сенсорів або інших периферійних пристроїв. Додатково програмний застосунок може

використовуватися для моделювання потоку та аналізу результатів роботи системи.

Наступним етапом є модуль попередньої обробки, який реалізує функціональність edge рівня. Саме тут виконуються основні операції над даними: фільтрація, агрегація та виявлення аномалій. Цей компонент є ключовим з точки зору всієї системи, оскільки саме він визначає, які дані будуть передані далі, а які будуть відкинуті.

Після обробки дані передаються через модуль, який відповідає за організацію потоку. Його задача полягає не лише у передачі інформації, але й у забезпеченні узгодженості між компонентами. У спрощеному вигляді він виконує роль зв'язуючого елемента між обробкою та прийомом даних.

Далі дані надходять до модуля прийому, який умовно представляє хмарний рівень. У межах цієї роботи він не виконує складних обчислень, а використовується для моделювання прийому даних і оцінки навантаження. Це дозволяє зосередитися на дослідженні попередньої обробки, не ускладнюючи систему зайвими деталями.

Окремо виділяється підсистема збору метрик. Вона взаємодіє з кількома компонентами одночасно та фіксує ключові показники роботи системи. Це дозволяє отримувати узагальнену картину того, що відбувається під час обробки даних, і оцінювати ефективність різних режимів роботи.

Не менш важливим елементом є інтерфейс користувача програмного застосунку. Хоча він не бере безпосередньої участі в апаратному зчитуванні даних, саме через нього здійснюється спостереження за роботою системи, перегляд метрик і порівняння результатів у різних режимах обробки. У контексті демонстраційної моделі це має особливе значення, оскільки результати повинні бути зрозумілими не лише з технічної, але й з візуальної точки зору.

Важливою особливістю розробленої архітектури є її модульність. Кожен компонент виконує обмежений набір функцій і не залежить від внутрішньої

реалізації інших частин системи. Це дозволяє змінювати або розширювати окремі модулі без суттєвого впливу на загальну структуру.

Ще один момент, який варто враховувати, полягає у підтримці різних режимів роботи. Архітектура побудована таким чином, що система може працювати як із попередньою обробкою, так і без неї. Це досягається за рахунок можливості змінювати маршрут проходження даних. У результаті стає можливим безпосередньо порівнювати два підходи в межах однієї системи.

Окрім цього, у процесі проектування було враховано необхідність асинхронного виконання. Потік даних обробляється незалежно від інтерфейсу користувача, що дозволяє уникнути блокувань і забезпечує більш плавну роботу системи. Це особливо важливо при високій інтенсивності надходження подій.

У підсумку можна сказати, що запропонована архітектура поєднує простоту реалізації з достатнім рівнем деталізації. Вона дозволяє чітко виділити основні етапи обробки даних і водночас залишає можливість для подальшого розвитку системи.

2.3 Моделювання потоків даних та сценаріїв їх надходження

При розробці системи попередньої обробки важливим є не лише визначення її структури, але й розуміння того, як саме поведуться дані в процесі роботи. Тобто мова йде не просто про компоненти, а про динаміку, а саме про те, як події з'являються, як проходять через систему і як змінюється їхній стан.

У даній роботі моделювання потоків даних розглядається як окремий етап, який дозволяє перевірити поведінку системи ще до оцінювання результатів. Це особливо важливо для потокових сценаріїв, де обробка відбувається безперервно і не має чітко визначених меж.

Основою моделювання є формування подій, які імітують роботу сенсорів або інших джерел телеметричних даних. У програмно-апаратній частині роботи таким джерелом виступає потенціометр, значення якого зчитується апаратним

					КвРКІ.2304276.22.04.85 ПЗ	Арк.
						32
Зм.	Арк.	№ докум.	Підпис	Дата		

edge-вузлом на базі ESP32. Додатково у програмному застосунку використовується модель генерації подій, що дозволяє змінювати параметри потоку та перевіряти різні режими роботи системи.

При цьому важливо, що події генеруються не одинично, а у вигляді потоку. Інтенсивність потоку може змінюватися, що дозволяє досліджувати поведінку системи при різному навантаженні. У реальних умовах це відповідає ситуаціям, коли кількість активних пристроїв або частота передачі даних змінюється.

Окрему увагу приділено моделюванню різних типів даних. Потік не є повністю однорідним: частина значень знаходиться в нормальному діапазоні, але періодично з'являються відхилення. Це зроблено спеціально, щоб продемонструвати роботу механізмів фільтрації та виявлення аномалій.

Крім того, у процесі моделювання враховується фактор часу. Події надходять із певним інтервалом, що дозволяє системі обробляти їх поступово. Такий підхід робить поведінку системи більш передбачуваною і дозволяє краще контролювати навантаження.

Для більш повного аналізу було передбачено кілька сценаріїв надходження даних. Перший сценарій відповідає рівномірному потоку, коли події генеруються з приблизно однаковою частотою. Це базовий випадок, який дозволяє оцінити стабільність роботи системи.

Другий сценарій передбачає змінне навантаження. У цьому випадку інтенсивність потоку може різко збільшуватися або зменшуватися. Такий підхід дозволяє перевірити, як система реагує на пікові значення та чи здатна вона підтримувати стабільну роботу.

Третій сценарій пов'язаний із появою аномальних значень. Тут основна увага приділяється не кількості подій, а їх змісту. Це дозволяє оцінити ефективність механізмів виявлення відхилень та їх вплив на загальний потік даних.

На рисунку 2.3 представлені основні сценарії надходження поточкових даних у систему.

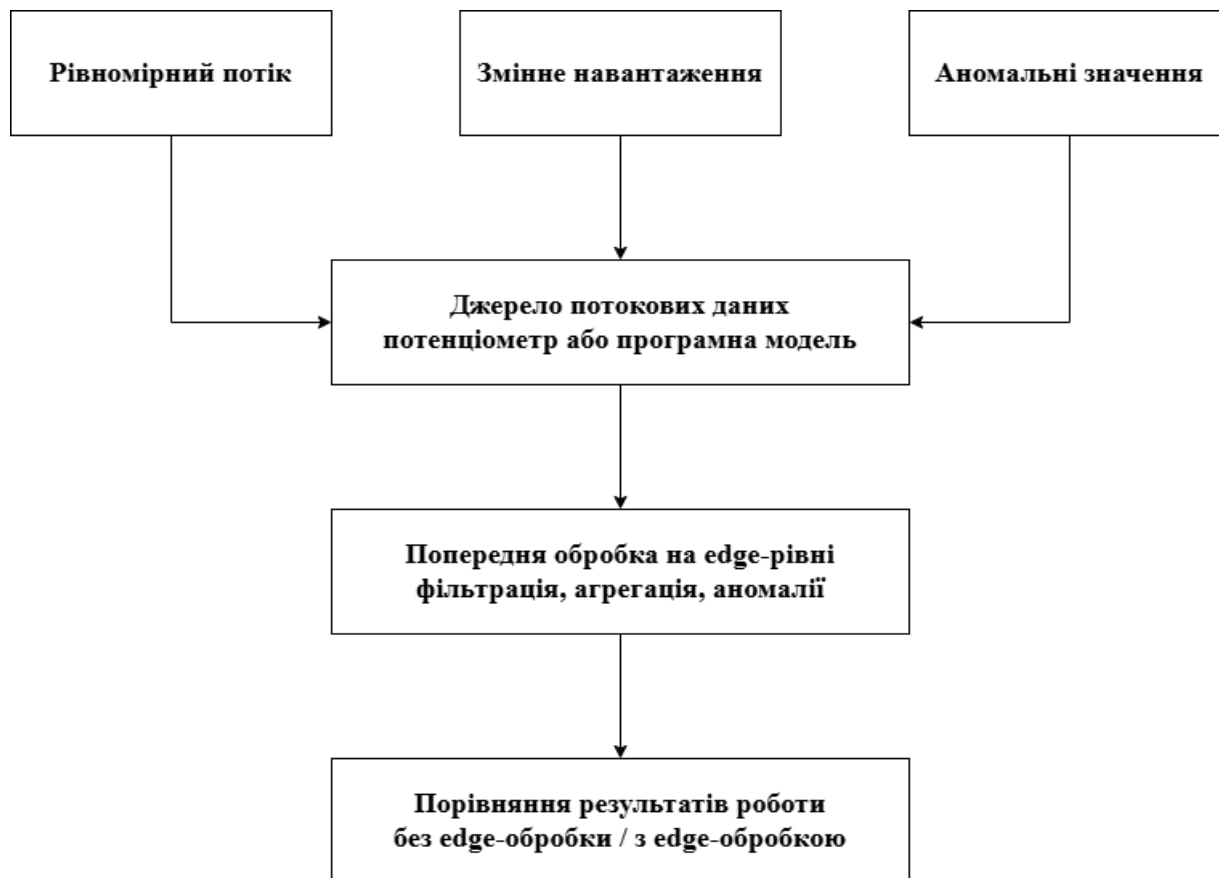


Рисунок 2.3 – Схема сценаріїв надходження поточкових даних

На схемі показано три основні сценарії надходження поточкових даних у систему. Перший сценарій відповідає рівномірному потоку, другий передбачає змінне навантаження, а третій пов'язаний із появою аномальних значень. Таке подання дозволяє розглянути роботу системи в різних умовах.

У кожному зі сценаріїв дані надходять від джерела поточкових даних, яким може бути потенціометр або програмна модель. Після цього вони проходять етап попередньої обробки на edge-рівні, де виконуються фільтрація, агрегація та виявлення аномалій.

Залежно від обраного режиму система може працювати без edge-обробки або з використанням попередньої обробки. Це дозволяє порівняти результати роботи в різних режимах і визначити, як локальна обробка впливає на обсяг переданих даних, затримку та стабільність роботи системи.

2.4 Проектування механізмів фільтрації, агрегації та трансформації даних

Після визначення структури системи та моделювання потоків даних наступним кроком є проектування механізмів їх обробки. Саме на цьому етапі формується логіка, яка безпосередньо впливає на те, які дані залишаються в системі, а які відсіюються. У контексті даної роботи основна увага приділяється трьом типам операцій: фільтрації, агрегації та виявленню аномалій. Трансформація даних розглядається як етап формування результату після попередньої обробки.

Ці механізми не існують окремо один від одного. Навпаки, вони працюють послідовно, і результат кожного етапу впливає на наступний. Через це важливо не тільки визначити самі алгоритми, але й зрозуміти порядок їх застосування. У програмно-апаратній системі ці операції виконуються на апаратному edge-вузлі ESP32 за допомогою Arduino-програми, а також відтворюються у програмному застосунку для моделювання та візуалізації роботи системи.

Першим етапом обробки є фільтрація даних. Вона виконується одразу після отримання події і має на меті відсіяти некоректні або шумові значення. У найпростішому випадку це перевірка на належність значення до допустимого діапазону. Такий підхід виглядає досить очевидним, але на практиці він дозволяє суттєво зменшити кількість даних, які потрапляють на наступні етапи.

Особливість фільтрації полягає в тому, що вона повинна бути максимально швидкою. Оскільки ця операція виконується для кожної події, навіть незначне ускладнення логіки може вплинути на загальну продуктивність. Тому у межах даної системи використовується просте правило перевірки, яке не потребує значних обчислень.

Після фільтрації дані переходять до етапу агрегації. На відміну від попереднього етапу, тут уже враховується не лише поточне значення, але й попередні дані. Для цього використовується підхід ковзного вікна, у межах якого зберігається обмежена кількість останніх значень для кожного джерела.

На основі цього вікна обчислюються агреговані характеристики, такі як середнє значення, мінімум та максимум. Це дозволяє отримати більш узагальнене уявлення про поведінку потоку, не зберігаючи всю історію даних. Важливо, що розмір вікна є обмеженим, що дозволяє контролювати використання пам'яті.

Агрегація виконує одразу кілька функцій. З одного боку, вона зменшує обсяг даних, які потрібно передавати далі. З іншого боку, агрегація підвищує їхню інформативність. Наприклад, середнє значення може краще відображати загальний стан системи, ніж окремі короточасні коливання.

Наступним етапом є трансформація даних. У межах даної системи вона включає формування нового повідомлення, яке містить не лише сире значення, але й результати попередньої обробки. Це може бути агрегована інформація або ознака аномалії.

Окремо варто розглянути механізм виявлення аномалій. У даній роботі використовується простий пороговий підхід. Якщо значення перевищує заданий рівень, воно позначається як аномальне. Такий метод не є складним з точки зору реалізації, але дозволяє наочно продемонструвати принцип роботи системи.

Важливо, що аномалії обробляються дещо інакше, ніж звичайні дані. Вони не підлягають агрегації і передаються далі одразу після виявлення. Це дозволяє зберегти їхню актуальність і забезпечити швидку реакцію системи.

Ще одним аспектом є зменшення обсягу переданих даних. У межах системи не всі події передаються далі після обробки. Частина з них використовується лише для внутрішніх обчислень і не виходить за межі edge рівня. Це дозволяє суттєво знизити навантаження на мережу.

При проектуванні механізмів обробки також було враховано необхідність їх узгодженої роботи. Операції не повинні конфліктувати між собою або створювати зайві затримки. Саме тому вони організовані у вигляді послідовного конвеєра, де кожен етап отримує вже підготовлені дані.

Для кращого розуміння логіки роботи цих механізмів вони представлені у вигляді узагальненої схеми на рисунку 2.4.

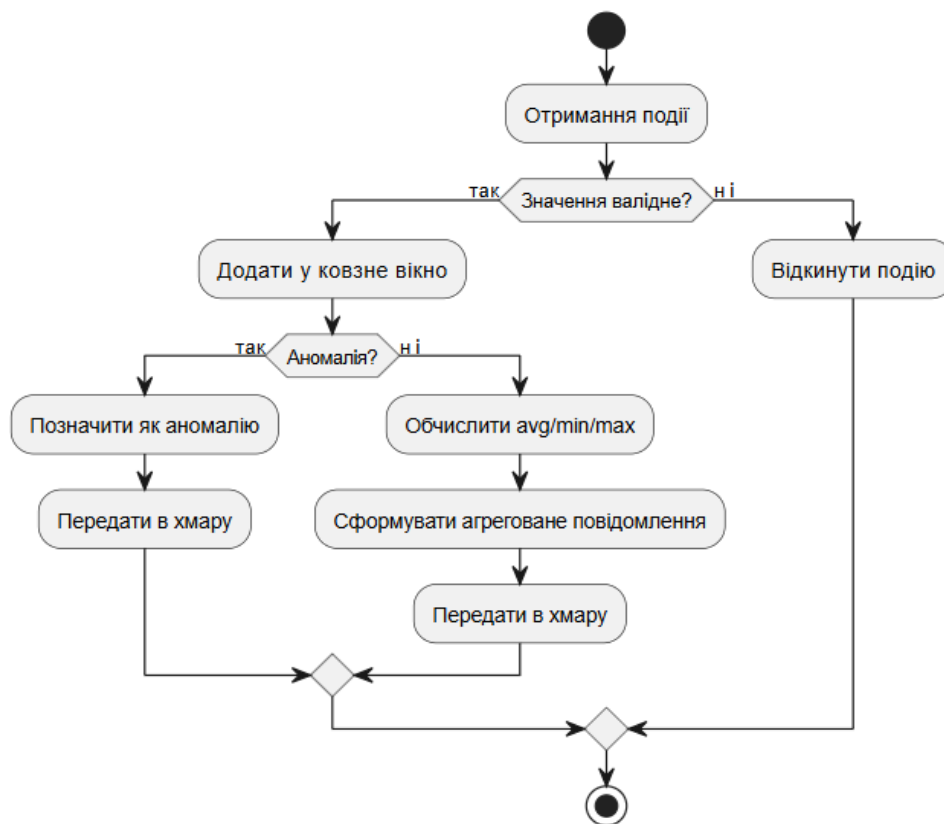


Рисунок 2.4 – Схема попередньої обробки даних на edge-рівні

На схемі показано послідовність проходження події через етапи обробки. Спочатку виконується перевірка коректності, після чого дані потрапляють у блок агрегації. Далі формується нове повідомлення, яке або передається далі, або відкидається.

Слід звернути увагу на те, що не всі події проходять через усі етапи. Наприклад, некоректні значення відсіюються одразу, а аномалії можуть обходити частину обробки. Це дозволяє оптимізувати роботу системи та уникнути зайвих обчислень.

У підсумку можна сказати, що запропоновані механізми обробки є досить простими, але водночас ефективними для демонстраційної системи. Вони

дозволяють показати основні принципи попередньої обробки даних без надмірного ускладнення логіки.

2.5 Вибір технологічного стеку

Після того як було визначено загальну архітектуру системи та принципи обробки даних, виникає практичне питання її реалізації. На цьому етапі необхідно обрати технології, які дозволять реалізувати задуману модель без зайвого ускладнення, але при цьому не обмежуватимуть функціональність.

Вибір технологічного стеку у даній роботі не був випадковим. Він формувався з урахуванням кількох факторів: типу задачі, вимог до продуктивності, необхідності візуалізації результатів, а також зручності розробки. Оскільки мова йде про демонстраційну систему, важливим було зберегти баланс між реалістичністю та простотою.

Оскільки розроблювана система має програмно-апаратний характер, технологічний стек доцільно розглядати у двох частинах: апаратній та програмній. Апаратна частина відповідає за імітацію надходження поточкових даних і їх локальну обробку на периферійному рівні, а програмна частина забезпечує моделювання, візуалізацію та порівняння результатів роботи системи.

Для реалізації апаратної частини використовується мікроконтролер ESP32. В межах даної роботи він виконує роль апаратного edge-вузла, який отримує вхідні значення з потенціометра, обробляє їх локально та формує результати роботи системи. Для керування режимами роботи використовується кнопка, а результати роботи апаратного вузла виводяться через Serial Monitor.

Оскільки система працює з поточковими даними, для програмної частини важливою є підтримка асинхронної обробки, яка дозволяє виконувати операції без блокування основного процесу виконання [58].

Для реалізації програмного застосунку обрано C#. Це рішення виглядає досить логічним, враховуючи, що мова добре підходить для створення настільних застосунків і має розвинені засоби роботи з асинхронністю. У контексті потокових даних це має значення, оскільки обробка повинна виконуватися без блокування основного потоку виконання. Крім того, C# дозволяє досить швидко реалізовувати подібні системи без необхідності використовувати складні сторонні інструменти [59].

Як середовище виконання використовується .NET Framework 4.8. На перший погляд це може виглядати не зовсім сучасним рішенням, однак у даному випадку такий вибір є виправданим. Платформа є стабільною, добре документованою і не потребує додаткового налаштування для запуску. Це важливо, оскільки система повинна без проблем працювати у стандартному середовищі, наприклад, під час демонстрації [60].

Окрему увагу було приділено вибору технології для створення інтерфейсу. Для цього використовується WPF, який дозволяє створювати достатньо гнучкі та візуально зрозумілі інтерфейси. У даному випадку це має не лише допоміжне, але й ключове значення, оскільки система повинна демонструвати результати обробки у наочному вигляді. Наявність графіків, логів та метрик значно спрощує сприйняття роботи системи [61].

Архітектурно інтерфейс побудований із використанням підходу MVVM. Це дозволяє відокремити логіку обробки даних від їх відображення. Такий поділ не лише спрощує структуру коду, але й робить систему більш зрозумілою. У результаті кожна частина виконує свою роль, і зміни в одному компоненті не потребують значних змін в інших [62].

Для реалізації обробки потокових даних у програмному застосунку не використовуються спеціалізовані платформи, як це часто робиться у промислових системах. Замість цього застосовуються вбудовані можливості мови, зокрема асинхронне програмування з використанням `async/await`. Це

дозволяє реалізувати неблокуючу обробку подій і підтримувати стабільну роботу інтерфейсу навіть при значній інтенсивності потоку [63].

Таке рішення має свої обмеження, однак у контексті даної роботи воно є достатнім. Основна мета полягає не в побудові масштабованої системи, а у демонстрації принципів обробки даних. Використання складних платформ, таких як Kafka або Flink, ускладнило б реалізацію і відволікло б від суті.

Саме тому було прийнято рішення реалізувати основні механізми обробки самостійно як у програмній частині, так і на рівні апаратного edge-вузла. Це дозволяє повністю контролювати логіку системи і чітко показати, як саме відбувається обробка даних на кожному етапі. Перевагою такого підходу є те, що система стає більш прозорою і зрозумілою з точки зору її роботи.

Окремо розглянуто роботу з метриками. У даній системі вони обчислюються без використання сторонніх бібліотек. Це зроблено для того, щоб уникнути зайвих залежностей і зберегти контроль над процесом обчислення. У результаті ключові показники можуть формуватися як в межах програмного застосунку, так і під час роботи апаратного edge-вузла.

Також було враховано, що система повинна працювати у реальному часі або близькому до нього. Це накладає певні обмеження на вибір технологій. Зокрема, необхідно уникати підходів, які можуть призводити до накопичення затримок або блокування виконання.

Ще одним фактором, який вплинув на вибір стеку, є можливість швидкого внесення змін. У процесі розробки демонстраційної системи часто виникає потреба змінювати логіку або параметри. Використання відносно простих технологій дозволяє робити це без значних витрат часу.

У підсумку обраний технологічний стек можна охарактеризувати як практичний для створення програмно-апаратної демонстраційної системи. Він не претендує на універсальність, але повністю відповідає поставленій задачі. Використані інструменти дозволяють реалізувати апаратний edge-вузол,

програмний застосунок, обробку поточкових даних і наочне відображення результатів роботи системи.

Такий підхід добре підходить для демонстраційних і дослідницьких задач, де важливо не лише отримати результат, але й показати, як саме він досягається.

2.6 Опис взаємодії компонентів та обміну повідомленнями

Після визначення архітектури системи та окремих механізмів обробки даних доцільно детальніше розглянути, як саме взаємодіють між собою її компоненти. На цьому етапі увага зміщується з внутрішньої логіки окремих модулів на обмін даними між ними. Саме ця взаємодія визначає цілісність системи та узгодженість її роботи.

У межах розробленої системи взаємодія компонентів організована за принципом послідовного проходження потоку даних. При цьому кожен компонент отримує дані, виконує певні дії та передає результат далі. Такий підхід дозволяє уникнути складних залежностей і зробити структуру системи більш прозорою.

Початковою точкою взаємодії є джерело поточкових даних. У межах розробленої системи цю роль може виконувати потенціометр або програмна модель, які формують вхідні значення та передають їх до наступного компонента без додаткової обробки. Такий підхід дозволяє поєднати апаратну частину системи з програмним моделюванням потоку.

Після цього дані надходять до апаратного edge-вузла або програмної частини системи, де визначається порядок їх подальшої обробки. Залежно від обраного режиму, подія або проходить попередню edge-обробку, або передається далі без додаткового скорочення. Це дозволяє реалізувати різні сценарії роботи без зміни загальної структури системи.

У випадку використання edge-обробки подія проходить послідовність операцій, серед яких фільтрація, агрегація та виявлення аномалій. Результатом

цього процесу є повідомлення, яке містить не лише поточне значення, але й додаткову службову інформацію про режим роботи, ознаку аномалії, агрегований показник і статус передачі.

Після завершення обробки або у випадку обходу edge-рівня дані передаються далі у вигляді повідомлення потокових даних. Таке повідомлення використовується для відображення результатів, порівняння режимів роботи та фіксації основних показників системи.

Ще одним важливим аспектом є мінімізація жорстких залежностей між компонентами. Кожен модуль взаємодіє з іншими через прості інтерфейси або узагальнені структури даних. Це дозволяє змінювати реалізацію окремих частин без необхідності перебудови всієї системи.

Для опису обміну даними між компонентами системи доцільно визначити структуру повідомлення, яке формується під час роботи. Основні складові такого повідомлення показано на рисунку 2.5.

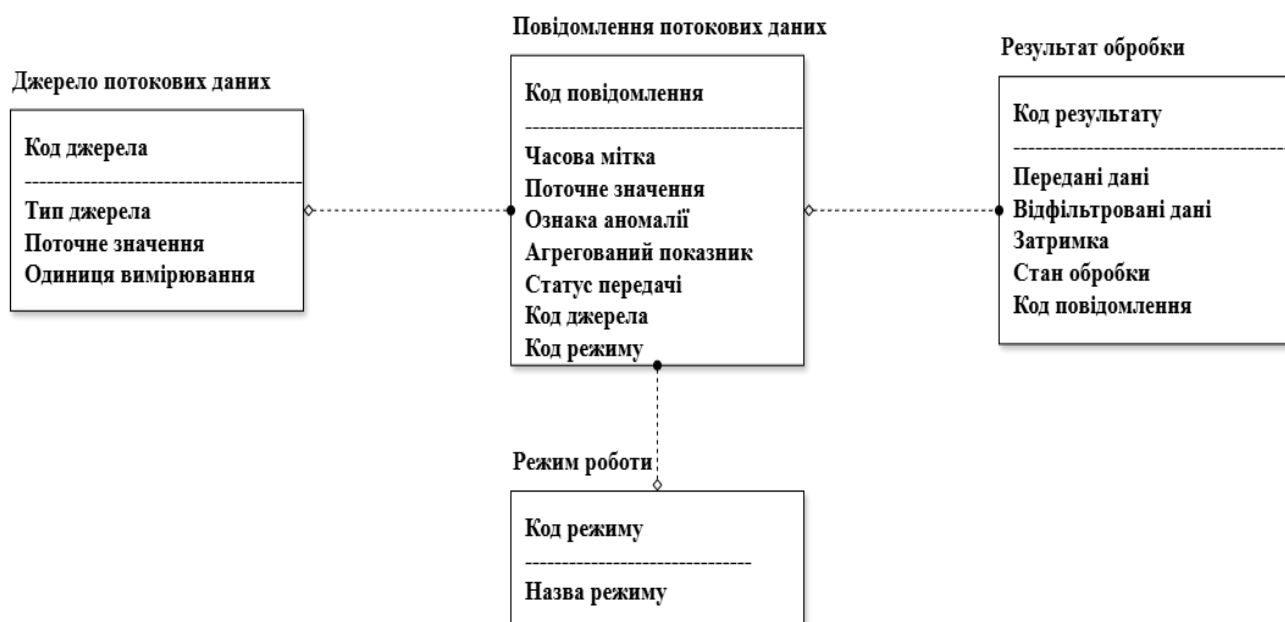


Рисунок 2.5 – Структура повідомлення потокових даних

На рисунку 2.5 показано структуру повідомлення потокових даних, яке використовується під час взаємодії компонентів системи. Центральним

елементом є повідомлення потокових даних, що містить часову мітку, поточне значення, режим роботи, ознаку аномалії, агрегований показник і статус передачі.

Окремо в структурі виділено джерело потокових даних, режим роботи та результат обробки. Це дозволяє передавати не лише саме значення, але й додаткову інформацію, яка потрібна для попередньої обробки, відображення результатів і порівняння режимів роботи системи.

У підсумку організація взаємодії компонентів забезпечує узгоджену роботу системи та дозволяє ефективно обробляти потокові дані. Використання єдиної структури повідомлення спрощує передачу даних між компонентами, зменшує залежність між окремими частинами системи та робить її більш зручною для подальшої реалізації.

2.7 Висновки до другого розділу

У другому розділі було розглянуто процес проєктування та моделювання програмно-апаратної системи периферійних обчислень для попередньої обробки потокових даних. Сформовано загальну концепцію побудови системи, яка передбачає розподіл процесу обробки на окремі логічні етапи та використання edge-рівня для виконання первинної обробки даних.

У ході роботи було визначено архітектуру системи, основні компоненти та особливості їх взаємодії. Система включає апаратний edge-вузол на базі ESP32, потенціометр як джерело потокових даних, кнопку для перемикання режимів роботи, а також програмний застосунок для моделювання, візуалізації та порівняння результатів. Встановлено, що модульний підхід дозволяє забезпечити гнучкість системи, спростити її структуру та створити можливість порівняння різних режимів обробки даних в межах єдиної моделі.

Також виконано моделювання потоків даних та сценаріїв їх надходження. Було передбачено різні режими формування подій, зокрема рівномірний потік,

змінне навантаження та появу аномальних значень. Це дозволило створити умови для подальшої перевірки поведінки системи при різних параметрах роботи.

У процесі проєктування визначено механізми фільтрації, агрегації та виявлення аномалій. Встановлено, що використання попередньої обробки на edge-рівні дозволяє зменшити обсяг переданих даних та підвищити їх інформативність.

Крім того, було визначено технологічний стек для реалізації системи. Для апаратної частини використовується ESP32 та Arduino-програма, а для програмної частини обрано мову програмування C#, платформу .NET Framework та технологію WPF для створення демонстраційного застосунку з візуальним інтерфейсом і підтримкою асинхронної обробки поточкових даних.

У результаті розглянуто особливості взаємодії компонентів системи та організацію обміну повідомленнями між ними. Запропонована структура повідомлення поточкових даних дозволяє узгодити передачу даних між компонентами, підтримує різні режими обробки та створює основу для подальшої програмно-апаратної реалізації системи.

3 ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ СИСТЕМИ ПЕРИФЕРІЙНИХ ОБЧИСЛЕНЬ ДЛЯ ПОПЕРЕДНЬОЇ ОБРОБКИ ПОТОКОВИХ ДАНИХ

3.1 Реалізація модулів збору та прийому поточкових даних

На етапі програмної реалізації особливу увагу було приділено модулям, що відповідають за формування та прийом поточкових даних. Саме вони визначають, як виглядає потік у системі та як він поводить себе під час виконання. У даній роботі ці функції реалізовані у вигляді окремих компонентів, що дозволяє чітко розділити генерацію та прийом даних.

В межах розробленої системи формування потоку даних розглядається у двох варіантах: за допомогою апаратного edge-вузла на базі ESP32 та за допомогою програмного генератора подій у C# застосунку. Апаратна частина використовується для імітації надходження даних від фізичного джерела, а програмна частина забезпечує моделювання потоку та подальше порівняння режимів роботи системи.

Модуль збору даних у межах програмної частини представлений генератором подій, який виконує роль імітаційного середовища. Паралельно з цим у роботі використовується апаратний edge-вузол на базі ESP32, у якому потенціометр формує змінне вхідне значення. Такий підхід дозволяє поєднати програмне моделювання з апаратною імітацією надходження поточкових даних.

Для реалізації апаратної частини було використано мінімальний набір компонентів, достатній для демонстрації принципу роботи edge-вузла. Потенціометр формує змінне вхідне значення, яке імітує надходження поточкових даних, а кнопка використовується для перемикання режимів роботи системи. Такий підхід дозволяє не ускладнювати апаратну частину і водночас перевірити основну логіку локальної обробки даних.

Схему підключення апаратного edge-вузла на базі ESP32 показано на рисунку 3.1.

					КвРКІ.2304276.22.04.85 ПЗ	Арк.
						45
Зм.	Арк.	№ докум.	Підпис	Дата		

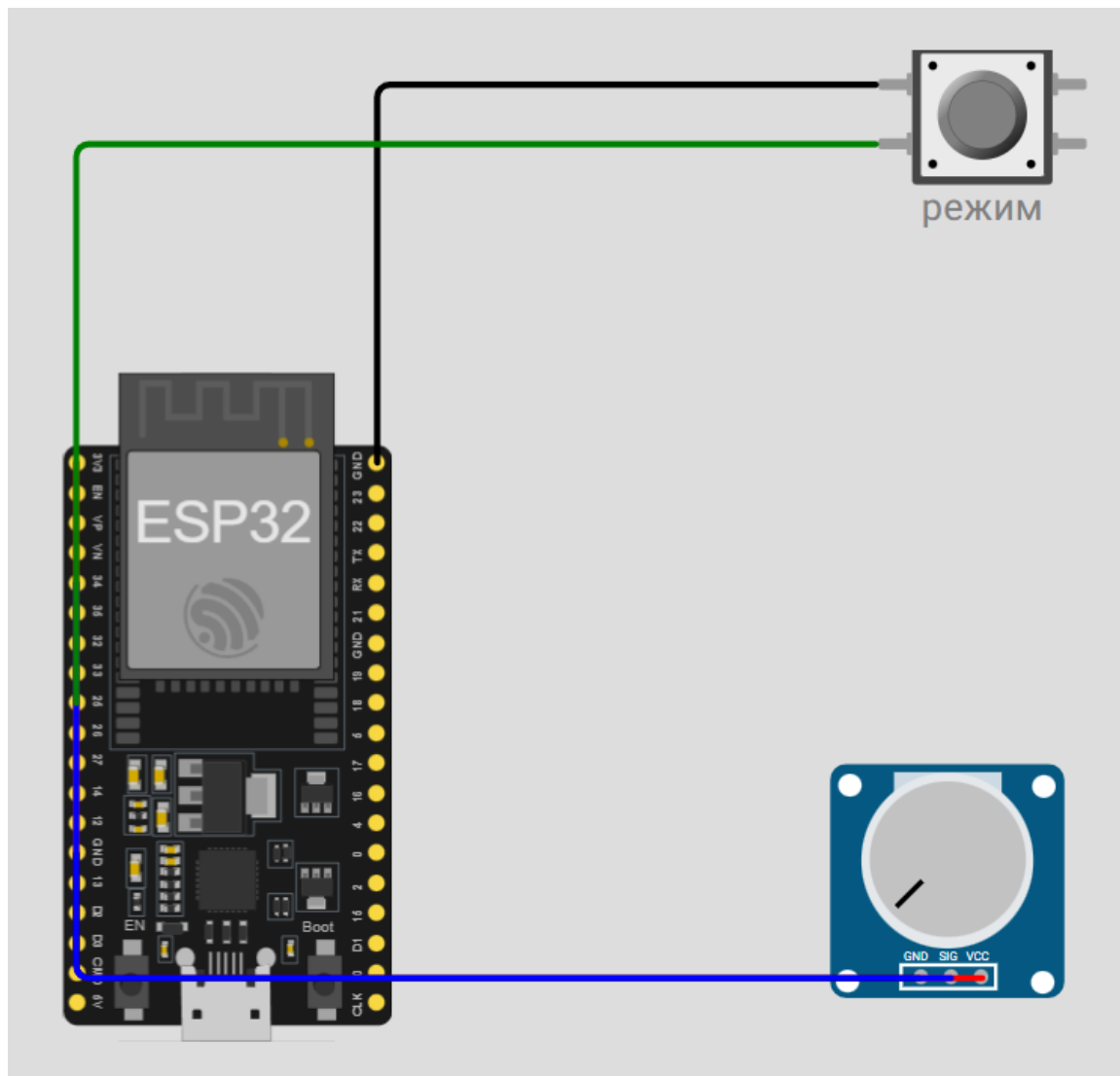


Рисунок 3.1 – Схема підключення апаратного edge-вузла на базі ESP32

На схемі показано підключення ESP32, потенціометра та кнопки. Потенціометр використовується як джерело змінного вхідного сигналу, а кнопка забезпечує перемикання режимів роботи системи. Отримані значення обробляються програмою мікроконтролера та можуть виводитися у Serial Monitor.

Основною задачею генератора є створення послідовності подій, які мають структуру, подібну до реальних телеметричних даних. Кожна подія містить часову мітку, ідентифікатор джерела та числове значення. Формування таких подій відбувається циклічно, що створює ефект безперервного потоку.

При реалізації генератора було враховано необхідність варіювання параметрів. Зокрема, передбачено можливість зміни швидкості генерації подій. Це дозволяє досліджувати поведінку системи при різному навантаженні. У практичному плані це виглядає як зміна кількості подій, що генеруються за одиницю часу.

Окрім цього, у процесі генерації враховується неоднорідність даних. Значення формуються навколо певного діапазону, однак періодично з'являються відхилення. Це дозволяє створити умови, у яких механізми обробки можуть проявити себе, зокрема в частині фільтрації та виявлення аномалій.

Важливою особливістю реалізації є використання асинхронного підходу. Генерація подій виконується у фоновому режимі і не блокує інші частини системи. Це дозволяє підтримувати стабільну роботу інтерфейсу користувача навіть при високій інтенсивності потоку.

Після формування у програмному застосунку події передаються до наступного модуля системи. На цьому етапі генератор не виконує жодної обробки і не змінює структуру даних. Його роль обмежується створенням потоку та передачею його далі. Такий підхід дозволяє зберегти чітке розмежування відповідальності між компонентами.

Другим важливим елементом є модуль прийому даних, який у даній системі виконує роль умовного хмарного рівня. Його основна функція полягає у фіксації отриманих повідомлень та моделюванні процесу їх обробки на стороні сервера.

На відміну від генератора, цей модуль працює вже з обробленими або частково обробленими даними. Він отримує повідомлення з потоку і виконує прості дії, пов'язані з їх обліком. У межах демонстраційної моделі цього достатньо, оскільки основна увага зосереджена на попередній обробці.

Окремо варто відзначити, що модуль прийому також використовується для оцінки затримок. Для цього у ньому реалізовано простий механізм імітації обробки, який дозволяє отримати приблизне значення часу між відправкою та

прийомом даних. Це дає змогу оцінити вплив різних режимів роботи системи на затримку.

Ще однією функцією модуля є передача інформації до підсистеми метрик. Кожне отримане повідомлення враховується при обчисленні загальних показників. Це дозволяє отримати більш точну картину роботи системи без необхідності додаткових обчислень.

Важливо, що модуль прийому не впливає на логіку обробки даних. Він не змінює структуру повідомлень і не виконує додаткових трансформацій. Це зроблено свідомо, щоб не зміщувати акцент з edge рівня на інші частини системи.

У процесі реалізації було також враховано необхідність узгодженої роботи модулів збору та прийому. Дані повинні передаватися без втрат, а система має коректно реагувати на зміну інтенсивності потоку. Для цього використовується послідовна передача подій із мінімальними затримками між етапами.

Події формуються генератором і передаються до модуля прийому через проміжні компоненти. Видно, що генератор не залежить від інших частин системи, а прийом даних відбувається після проходження всіх етапів обробки.

Слід звернути увагу на те, що обидва модулі працюють незалежно, але узгоджено. Генератор створює потік, а модуль прийому його обробляє. Між ними немає жорсткої прив'язки, що дозволяє змінювати окремі частини системи без порушення загальної логіки.

У підсумку реалізація модулів збору та прийому даних створює основу для роботи всієї системи. Вона забезпечує формування потоку даних як на рівні апаратного вузла, так і в межах програмного застосунку, його передачу та фіксацію результатів, що є необхідним для подальшого оцінювання ефективності обробки.

3.2 Реалізація підсистеми попередньої обробки на edge-рівні

Ключовим елементом розробленої системи є підсистема попередньої обробки даних, яка реалізує функціональність периферійного рівня. Саме на цьому етапі відбувається основна трансформація потоку, що безпосередньо впливає на обсяг переданих даних і загальну ефективність системи.

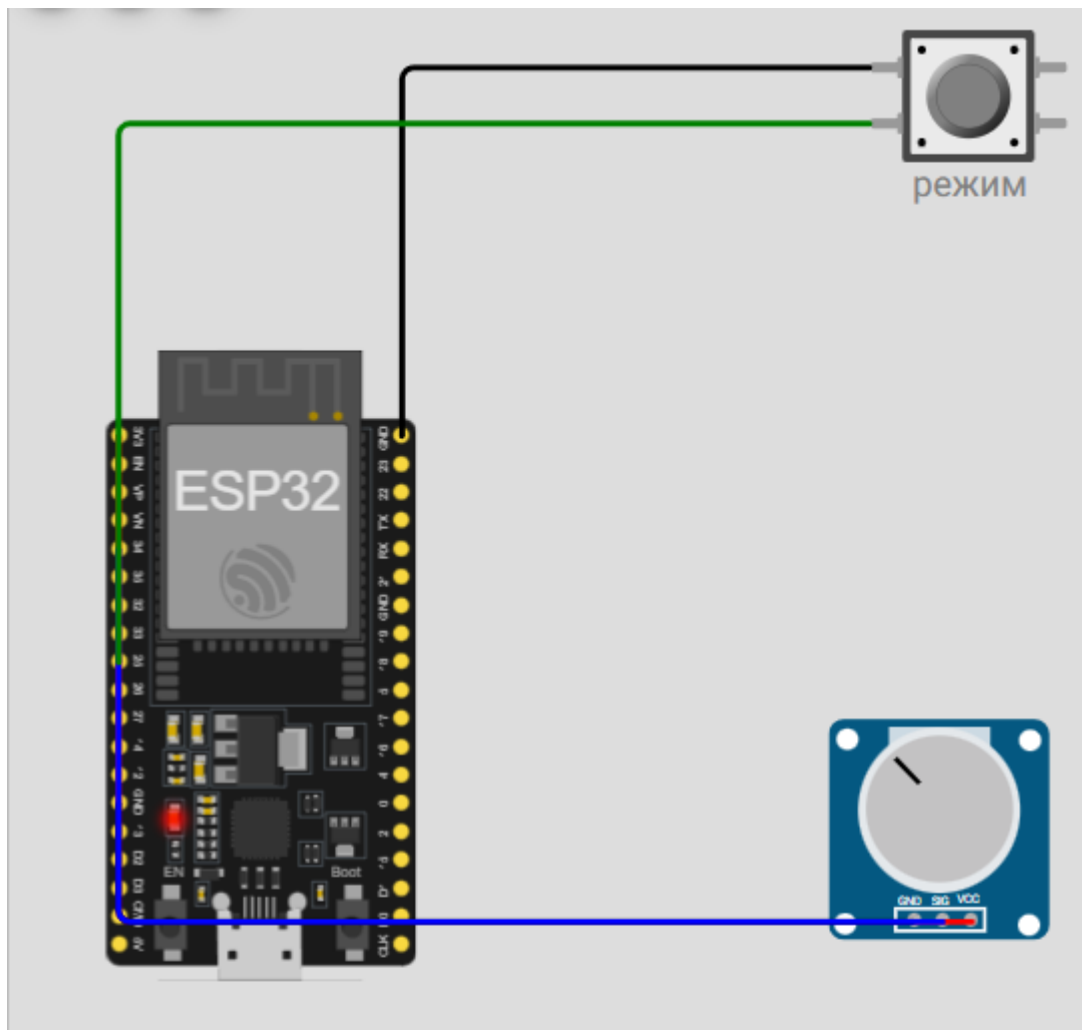
В межах програмно-апаратної реалізації edge-рівень представлений апаратним вузлом на базі ESP32 та відповідним модулем у програмному застосунку. Arduino-програма на ESP32 виконує локальне зчитування значень, перевірку режиму роботи, фільтрацію, агрегацію та виявлення аномалій, а програмний застосунок відтворює ці механізми для моделювання та порівняння результатів.

На відміну від інших компонентів, які виконують допоміжні функції, edge рівень визначає, які дані залишаються в системі, а які передаються далі. У межах даної роботи ця підсистема реалізована у вигляді окремого програмного модуля та логіки мікроконтролера, які приймають на вхід сирі значення або події та повертають результат їх обробки.

Обробка організована як послідовність відносно простих операцій. Кожна подія проходить через кілька етапів, і на кожному з них визначається її подальша обробка. Такий підхід дозволяє уникнути надмірної складності та робить логіку роботи більш зрозумілою.

У межах апаратної частини така обробка виконується безпосередньо програмою мікроконтролера. Це дозволяє не передавати всі отримані значення далі, а спочатку перевірити їх на рівні edge-вузла. У результаті частина даних може бути відфільтрована або використана для обчислення узагальнених показників ще до передачі на наступні етапи системи.

Приклад виведення результатів роботи програми мікроконтролера у Serial Monitor показано на рисунку 3.2.



Комп'ютерна система периферійних обчислень

Поточний режим: з edge

Поточне значення потоку: 337

Ковзне середнє: 337.0

Стан потоку: нормальний

Передача до хмари: ні

Причина передачі: дані відфільтровано на edge-вузлі

Кількість отриманих повідомлень: 18

Кількість переданих повідомлень: 4

Кількість відфільтрованих повідомлень: 14

Зменшення трафіку: 77.8 %

Рисунок 3.2 – Виведення результатів роботи програми мікроконтролера у Serial Monitor

На рисунку 3.2 показано приклад роботи програми мікроконтролера у режимі з edge-обробкою. У Serial Monitor відображаються поточне значення потоку, ковзне середнє, стан потоку, рішення щодо передачі даних, а також кількість отриманих, переданих і відфільтрованих повідомлень. Це підтверджує, що частина попередньої обробки виконується безпосередньо на рівні апаратного edge-вузла.

Першим етапом є фільтрація. Вона виконується одразу після отримання вхідного значення або події і полягає у перевірці значення на належність до допустимого діапазону. Якщо значення виходить за встановлені межі, подія не обробляється далі. Це дозволяє відсіяти некоректні або шумові дані ще на початковому рівні.

Після цього подія переходить до етапу агрегації. Тут використовується підхід ковзного вікна, у межах якого зберігається обмежена кількість останніх значень для кожного джерела. На основі цих даних обчислюються агреговані показники, зокрема середнє значення, мінімум та максимум.

Окремо реалізовано механізм виявлення аномалій. У даній системі використовується пороговий підхід: якщо значення перевищує заданий рівень, воно вважається аномальним. Такі події не накопичуються, а передаються далі одразу після виявлення.

Важливо, що не всі події проходять повний цикл обробки. Частина з них відсіюється ще на етапі фільтрації, а аномальні значення можуть обходити агрегацію. Це дозволяє зменшити кількість зайвих обчислень і оптимізувати роботу системи.

Обробка кожної події відбувається поетапно, із прийняттям рішень на ключових ділянках. Така організація дозволяє гнучко керувати потоком даних та забезпечує ефективне використання ресурсів.

Окрім загальної логіки, важливо також розглянути внутрішню структуру підсистеми. Вона включає кілька взаємопов'язаних компонентів, кожен з яких виконує окрему функцію.

Основний компонент координує роботу модулів фільтрації, агрегації та виявлення аномалій. При цьому агрегатор взаємодіє з механізмом ковзного вікна, який забезпечує збереження обмеженої кількості попередніх значень.

Слід звернути увагу на те, що всі елементи підсистеми є відносно простими та виконують чітко визначені функції. Це дозволяє уникнути зайвої складності і водночас забезпечує достатній рівень функціональності.

У підсумку підсистема попередньої обробки виконує роль центрального елемента всієї системи. Вона забезпечує зменшення обсягу даних, виявлення аномалій та формування більш інформативних повідомлень, що дозволяє ефективно реалізувати підхід периферійних обчислень у межах розробленої програмно-апаратної системи.

3.3 Реалізація механізмів фільтрації, агрегації та виявлення аномалій

Після реалізації підсистеми попередньої обробки логічним продовженням є детальний розгляд механізмів, які безпосередньо забезпечують перетворення поточкових даних. Якщо у попередньому пункті увага приділялася загальній структурі та організації обробки, то в даному випадку доцільно зосередитися на внутрішній логіці окремих операцій та їх узгодженій роботі.

У даному підрозділі механізми обробки розглядаються як спільна логіка, що використовується під час роботи апаратного edge-вузла та програмного застосунку. Це дозволяє зберегти єдиний принцип обробки поточкових даних у двох частинах системи.

У межах розробленої програмно-апаратної системи механізми обробки побудовані за принципом послідовного конвеєра. Це означає, що кожна подія проходить через визначений набір етапів, причому порядок їх виконання є фіксованим. Такий підхід дозволяє уникнути складних умовних переходів і спрощує аналіз поведінки системи.

Водночас важливо підкреслити, що конвеєр не є жорстко лінійним у класичному розумінні. На окремих етапах можливе прийняття рішень, які впливають на подальший маршрут події. Таким чином, структура обробки поєднує елементи послідовного та умовного виконання.

Першим механізмом, який застосовується до даних, є фільтрація. Його основне завдання полягає у відсіюванні некоректних значень ще до початку більш складних обчислень. У контексті даної системи фільтрація реалізована як перевірка значення на відповідність заданому діапазону. Це дозволяє уникнути обробки шумових даних, які не мають практичної цінності.

На перший погляд така перевірка виглядає елементарною, однак її роль у системі є суттєвою. Саме вона визначає, який обсяг даних потрапить на наступні етапи. Якщо фільтрація налаштована надто жорстко, система може втратити корисну інформацію. Якщо ж вона занадто м'яка, зростає навантаження на інші механізми.

Після проходження етапу фільтрації подія передається до механізму агрегації. На цьому етапі змінюється сам характер обробки: система починає враховувати не лише поточне значення, але й попередні дані. Для цього використовується структура ковзного вікна, яка зберігає обмежену кількість останніх значень.

Використання ковзного вікна дозволяє досягти балансу між точністю та ефективністю. З одного боку, система отримує можливість аналізувати динаміку змін. З іншого боку, вона не потребує збереження повної історії даних, що могло б призвести до надмірного використання пам'яті.

На основі даних, що знаходяться у вікні, обчислюються агреговані показники. Найбільш очевидними з них є середнє значення, мінімум та максимум. Однак навіть ці прості характеристики дозволяють отримати більш повне уявлення про стан потоку, ніж окремі значення.

Слід зазначити, що агрегація виконує не лише аналітичну, але й оптимізаційну функцію. Завдяки узагальненню даних зменшується кількість

повідомлень, які передаються далі. Це безпосередньо впливає на навантаження мережі та швидкість роботи системи.

Окрему роль у системі відіграє механізм виявлення аномалій. Його задача полягає у виявленні значень, які суттєво відрізняються від нормального діапазону. У даній реалізації використовується пороговий підхід, який базується на порівнянні значення з наперед визначеним рівнем.

Хоча такий підхід не можна назвати складним або універсальним, він має свої переваги. По-перше, він не потребує значних обчислювальних ресурсів. По-друге, його поведінка є передбачуваною, що важливо для демонстраційної системи. По-третє, він дозволяє чітко показати сам принцип виявлення відхилень.

Особливістю реалізації є те, що аномалії обробляються окремо від звичайних даних. Вони не включаються у процес агрегації і не накопичуються у ковзному вікні. Це дозволяє уникнути спотворення агрегованих показників і зберегти їхню коректність.

Ще одним важливим аспектом є формування вихідного повідомлення. Після завершення всіх етапів обробки система формує нову структуру даних, яка може містити як початкове значення, так і результати обчислень. Це дозволяє передавати більш інформативні дані без збільшення кількості повідомлень.

При цьому не всі події доходять до цього етапу. Частина з них відсіюється або використовується лише для внутрішніх розрахунків. Такий підхід дозволяє реалізувати механізм скорочення потоку даних, який є одним із ключових елементів периферійних обчислень.

Окремо варто звернути увагу на узгодженість роботи всіх механізмів. Вони не повинні працювати ізольовано або конфліктувати між собою. У розробленій системі це досягається за рахунок чіткої послідовності виконання та мінімізації взаємних залежностей.

Також було враховано фактор продуктивності. Усі механізми реалізовані таким чином, щоб їх виконання займало мінімальний час. Це особливо важливо, оскільки обробка відбувається для кожної події окремо.

У практичному сенсі це означає, що система здатна обробляти потік даних без суттєвих затримок навіть при зростанні інтенсивності. При цьому складність алгоритмів залишається на прийнятному рівні, що відповідає цілям даної роботи.

Під час проходження через основні етапи обробки подія поступово змінює свою структуру та доповнюється результатами попередньої обробки. Окремі механізми виконують свої функції незалежно, але водночас утворюють єдиний логічний процес.

Слід звернути увагу на те, що система не є жорстко фіксованою. У разі необхідності окремі механізми можуть бути змінені або доповнені. Це робить її придатною для подальшого розвитку.

У підсумку можна сказати, що розроблені механізми обробки забезпечують ефективне перетворення поточкових даних та дозволяють реалізувати основні принципи периферійних обчислень. Вони поєднують простоту реалізації з достатньою функціональністю, що є оптимальним для задач даного типу.

3.4 Інтеграція компонентів у єдину систему

Після реалізації окремих модулів та механізмів обробки наступним етапом є їх об'єднання в єдину систему. Саме на цьому рівні перевіряється не лише коректність роботи кожного компонента, але й узгодженість їх взаємодії. У результаті формується цілісна структура, у якій усі елементи працюють як єдине ціле.

Інтеграція у даній роботі розглядається не як формальне поєднання модулів, а як процес забезпечення стабільного обміну даними між ними.

Важливо, щоб кожен компонент отримував необхідні дані у потрібний момент і передавав результати без втрат або затримок.

У межах системи інтеграція здійснюється на рівні потоку даних. Усі компоненти підключені до спільного процесу обробки, у якому події послідовно проходять через визначені етапи. Такий підхід дозволяє уникнути складних залежностей і забезпечує прозорість роботи системи.

Початковою точкою інтеграції є джерело поточкових даних. У межах розробленої системи ним може бути апаратний edge-вузол на базі ESP32 або програмний генератор подій у C# застосунку. Апаратний вузол формує вхідні значення за допомогою потенціометра, а програмний генератор використовується для моделювання потоку та перевірки різних режимів роботи.

Далі, залежно від обраного режиму, дані або направляються до підсистеми попередньої обробки, або передаються безпосередньо до модуля прийому. Така організація дозволяє реалізувати кілька сценаріїв роботи в межах однієї системи без зміни її загальної структури.

Після обробки або у випадку обходу edge рівня дані надходять до модуля прийому. Цей компонент завершує основний цикл обробки та фіксує результати. Паралельно з цим відбувається передача інформації до підсистеми метрик.

Підсистема метрик інтегрована з кількома компонентами одночасно. Вона отримує дані як від програмного генератора, так і від модулів обробки, прийому та апаратного edge-вузла на базі ESP32. Це дозволяє формувати узагальнену інформацію про роботу системи без необхідності втручання в основний потік.

Окрему роль у процесі інтеграції відіграє інтерфейс користувача. Він не бере участі в обробці даних, але забезпечує керування системою та відображення результатів. Взаємодія з інтерфейсом відбувається через підсистему метрик та керуючі компоненти.

Важливою особливістю інтеграції є асинхронний характер взаємодії. Компоненти не блокують один одного, що дозволяє системі стабільно працювати

навіть при значному навантаженні. Це досягається за рахунок розділення потоків виконання та використання неблокуючих операцій.

Ще одним аспектом є відсутність жорсткої прив'язки між компонентами. Взаємодія здійснюється через узагальнені структури даних, що дозволяє змінювати внутрішню реалізацію окремих модулів без впливу на інші частини системи.

Більшість обчислень виконується на локальному вузлі, тоді як модуль прийому умовно представляє віддалене середовище.

Слід звернути увагу на те, що програмна частина системи реалізована як єдиний застосунок, тоді як апаратна частина представлена окремим edge-вузлом на базі ESP32. При цьому логічно система поділяється на кілька рівнів, що відповідають різним етапам обробки. Це дозволяє розглядати її як спрощену програмно-апаратну модель розподіленої системи.

У підсумку інтеграція компонентів забезпечує узгоджену роботу системи та дозволяє реалізувати всі передбачені сценарії обробки. Вона поєднує простоту реалізації з достатньою гнучкістю, що є важливим для демонстраційної моделі.

3.5 Проведення експериментів із поточковими даними

Після завершення реалізації системи виникає потреба перевірити, як вона поводить себе не в теорії, а під час реальної роботи. У цьому контексті експерименти виконують роль інструмента, який дозволяє побачити систему «в русі» – при різному навантаженні, у різних режимах і з різними типами даних.

У межах роботи перевірка виконувалася як для програмної частини системи, так і для апаратного edge-вузла на базі ESP32. Програмний застосунок використовувався для моделювання потоків даних і порівняння режимів, а апаратна частина дозволяла перевірити локальну обробку значень, отриманих із потенціометра.

Перед початком дослідження було визначено кілька параметрів, які впливають на поведінку системи. Насамперед це швидкість генерації подій. Зі збільшенням цього параметра система починає працювати в більш напруженому режимі, що дозволяє виявити потенційні обмеження. Окрім цього враховувалася кількість джерел даних, що впливає на структуру потоку.

Окрему роль відіграє режим обробки. У системі передбачено два варіанти: із використанням edge рівня та без нього. Перехід між ними не потребує зміни конфігурації компонентів, що дозволяє проводити порівняння у максимально однакових умовах.

Приклад зміни режиму роботи системи під час тестування показано на рисунку 3.3.

Комп'ютерна система периферійних обчислень	Комп'ютерна система периферійних обчислень
Поточний режим: з edge	Поточний режим: без edge
Поточне значення потоку: 555	Поточне значення потоку: 555
Ковзне середнє: 294.0	Ковзне середнє: 555.0
Стан потоку: аномалія	Стан потоку: нормальний
Передача до хмари: так	Передача до хмари: так
Причина передачі: виявлено аномалію	Причина передачі: усі дані передаються без обробки
Кількість отриманих повідомлень: 4	Кількість отриманих повідомлень: 1
Кількість переданих повідомлень: 3	Кількість переданих повідомлень: 1
Кількість відфільтрованих повідомлень: 1	Кількість відфільтрованих повідомлень: 0
Зменшення трафіку: 25.0 %	Зменшення трафіку: 0.0 %
Режим роботи змінено.	
Новий режим: без edge	
Статистику потоку очищено для нового режиму.	

Рисунок 3.3 – Зміна режиму роботи системи під час тестування

На рисунку 3.3 показано перемикання режимів роботи системи без edge-обробки та з використанням edge-обробки. Це дозволяє перевірити, як змінюється поведінка системи при прямій передачі даних і при виконанні попередньої обробки на периферійному рівні.

Перші спостереження були отримані під час роботи з відносно стабільним потоком. Події надходили рівномірно, без різких змін інтенсивності. У таких

умовах система працювала передбачувано, а різниця між режимами проявлялася переважно в обсязі переданих даних. Навіть без детальних вимірювань було помітно, що при використанні попередньої обробки частина подій не доходить до кінцевого вузла.

Далі умови було ускладнено шляхом поступового збільшення швидкості генерації. Це дозволило побачити, як система реагує на зростання навантаження. У режимі без edge обробки всі події передавалися далі, що призводило до помітного збільшення кількості повідомлень. У той же час при використанні попередньої обробки частина даних відсіювалася, і навантаження на наступні етапи зростало значно повільніше.

Цікавим виявився і сценарій із нерівномірним потоком. Коли інтенсивність змінюється не поступово, а стрибками, система поводить себе менш стабільно. У такі моменти стає помітно, що наявність попередньої обробки допомагає згладжувати навантаження. Дані не накопичуються в такій кількості, як у випадку прямої передачі.

Окремо варто згадати експерименти з аномальними значеннями. У цьому випадку система не лише обробляє потік, але й виділяє окремі події. Було помітно, що такі значення не «губляться» серед інших, а передаються далі практично одразу. Це підтверджує, що механізм виявлення аномалій працює незалежно від інших етапів обробки.

Під час експериментів фіксувалися кілька показників, але результати розглядалися не лише як набір числових значень. Наприклад, кількість переданих повідомлень дає уявлення про навантаження, однак сама по собі не пояснює причину змін. Тому результати розглядалися у взаємозв'язку.

Затримка обробки також оцінювалася не як окреме число, а як тенденція. При збільшенні навантаження вона зростає, але швидкість цього зростання відрізняється залежно від режиму роботи. Це дозволяє зробити висновок про вплив попередньої обробки на стабільність системи.

Якщо підсумувати отримані спостереження, можна сказати, що система поводить передбачувано і відповідає очікуванням. Попередня обробка дійсно впливає на обсяг переданих даних і дозволяє зменшити навантаження. При цьому загальна логіка роботи системи залишається стабільною навіть при зміні умов.

У результаті проведені експерименти створюють основу для подальшого розгляду продуктивності системи у наступному пункті.

3.6 Дослідження продуктивності та затримок обробки

Після проведення експериментів наступним кроком є аналіз отриманих результатів. На цьому етапі увага зосереджується не на самій організації системи, а на її поведінці під час роботи. Основною задачею є оцінка того, як змінюються ключові характеристики при використанні попередньої обробки даних.

У межах дослідження розглядалися три основні показники: кількість переданих повідомлень, обсяг даних та затримка обробки. Вони дозволяють оцінити ефективність системи з різних сторін – як з точки зору навантаження, так і з точки зору швидкодії.

Перш за все було проаналізовано кількість повідомлень, що передаються до модуля прийому. У режимі без використання попередньої обробки цей показник практично збігається з кількістю згенерованих подій. Наприклад, при інтенсивності потоку на рівні близько 500 подій за секунду до приймача надходила майже така ж кількість повідомлень. Це свідчить про відсутність будь-якої оптимізації на етапі передачі.

У разі використання edge обробки ситуація змінюється. Частина подій відсіюється під час фільтрації, а інша частина об'єднується у процесі агрегації. У результаті кількість переданих повідомлень зменшується приблизно до 180–

250 за секунду. Таким чином, потік скорочується майже вдвічі, що безпосередньо впливає на навантаження системи.

Аналогічна тенденція спостерігається і при оцінці обсягу даних. У класичному режимі обсяг переданої інформації зростає пропорційно кількості подій. У той же час при використанні попередньої обробки значна частина даних не передається або передається у вигляді узагальнених значень. У середньому це дозволяє зменшити обсяг трафіку у 2–2,5 рази.

Окремо було розглянуто затримку обробки. У режимі без попередньої обробки вона формується переважно за рахунок передачі даних і подальшої обробки. При помірному навантаженні середнє значення затримки становило близько 40–45 мс.

У режимі з використанням edge обробки ситуація є менш однозначною. З одного боку, додається додатковий етап обробки, що потенційно може збільшувати затримку. З іншого боку, зменшується обсяг переданих даних, що скорочує час передачі. У результаті середня затримка становила приблизно 25–30 мс, тобто залишалася на тому ж рівні або навіть дещо зменшувалася.

Особливо помітною різниця стає при зростанні навантаження. У випадку прямої передачі затримка починає збільшуватися швидше, оскільки система змушена обробляти більший обсяг даних. При використанні попередньої обробки цей процес відбувається більш плавно, що свідчить про кращу адаптацію до змін інтенсивності потоку.

Для більш наочного представлення результатів було використано візуалізацію, отриману безпосередньо з інтерфейсу системи.

Використання графічного відображення дозволяє краще простежити різницю між режимами роботи системи. У програмному застосунку відображаються сформовані події, кількість переданих повідомлень, показники затримки та загальний стан потоку. Це дає змогу не лише отримати числові значення, але й візуально оцінити, як змінюється навантаження при використанні попередньої обробки.

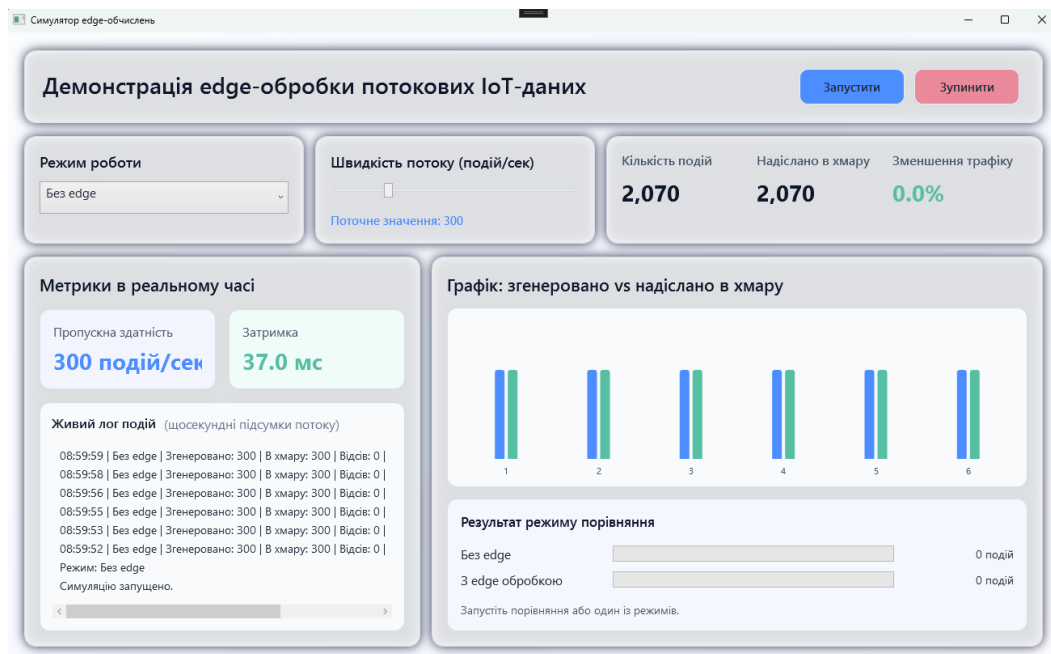


Рисунок 3.4 – Відображення потоку даних без використання edge обробки

На рисунку показано роботу системи у режимі прямої передачі. Видно, що кількість повідомлень, які надходять до модуля прийому, відповідає кількості згенерованих подій. Це створює підвищене навантаження на систему та призводить до збільшення затримок при зростанні інтенсивності потоку.

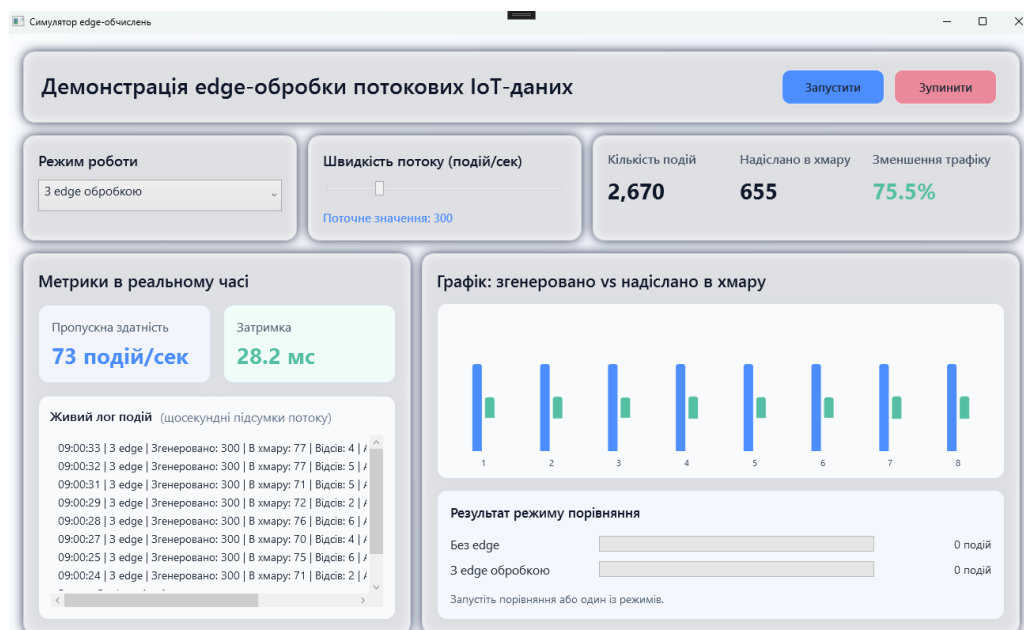


Рисунок 3.5 – Відображення потоку даних із використанням edge обробки

У цьому випадку спостерігається зменшення кількості переданих повідомлень. Частина даних відсіюється або агрегується, що дозволяє знизити навантаження на систему без втрати основної інформації. Потік виглядає більш «стабільним» навіть при зміні умов.

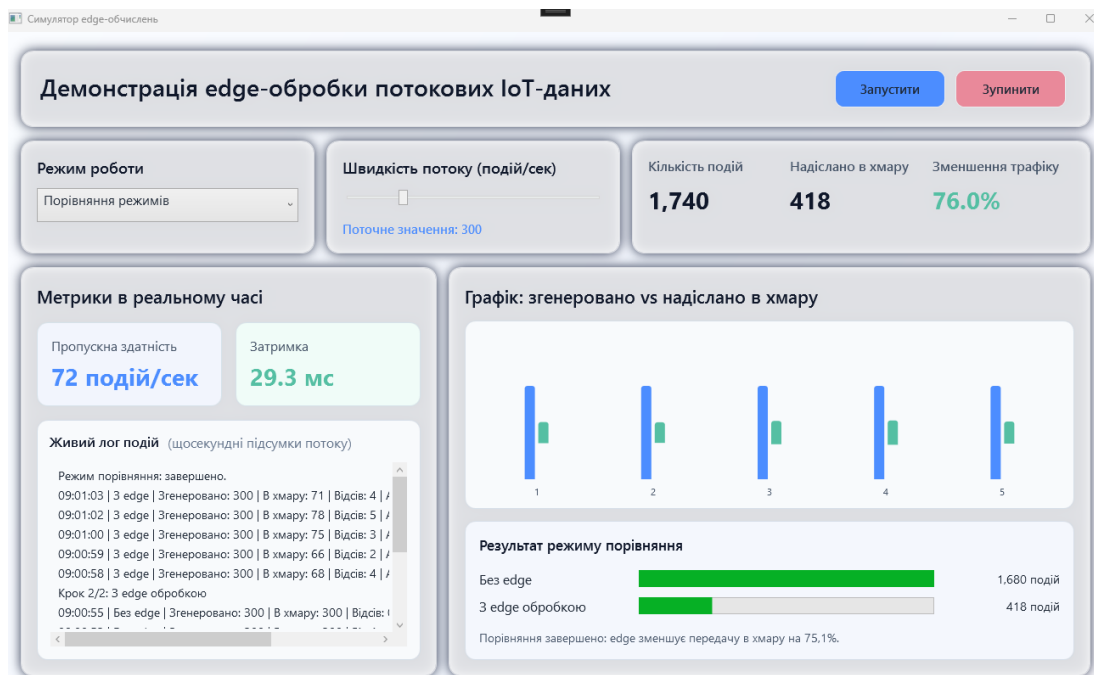


Рисунок 3.6 – Порівняння затримок обробки в різних режимах

З рисунка видно, що затримка у режимі з попередньою обробкою не перевищує затримку у класичному режимі, а в окремих випадках є меншою. Це пояснюється тим, що зменшення обсягу переданих даних компенсує витрати на локальну обробку.

Загалом отримані результати свідчать про те, що використання периферійної обробки дозволяє знизити навантаження на систему без негативного впливу на швидкодію. У деяких сценаріях це навіть призводить до покращення загальної продуктивності.

Таким чином, проведений аналіз підтверджує, що попередня обробка даних є доцільною у випадках роботи з інтенсивними потоками, де важливим є баланс між швидкістю та обсягом переданої інформації.

3.7 Оцінка ефективності запропонованого рішення

Після проведення експериментів та аналізу отриманих результатів доцільно узагальнити ефективність запропонованого підходу. На цьому етапі важливо не лише перелічити отримані показники, але й оцінити їх значення в контексті поставленої задачі. Оцінювання виконувалося з урахуванням роботи програмного застосунку та апаратного edge-вузла на базі ESP32, що дозволяє розглядати результати як для програмної моделі, так і для апаратної реалізації попередньої обробки.

Основним результатом роботи є підтвердження того, що використання попередньої обробки даних на периферійному рівні дозволяє змінити характер роботи системи. Найбільш помітний ефект спостерігається у зменшенні кількості переданих повідомлень та загального обсягу даних. У середньому обсяг переданої інформації зменшується приблизно у 2–2,5 рази, що безпосередньо впливає на навантаження мережі.

Не менш важливим є вплив на затримку обробки. У процесі експериментів було встановлено, що при використанні edge обробки середня затримка становить близько 25–30 мс, тоді як у класичному режимі вона знаходиться в межах 40–45 мс. Це свідчить про те, що попередня обробка не лише не погіршує швидкодію системи, але й у певних умовах дозволяє її покращити.

У відносному вираженні це відповідає скороченню затримки приблизно на 20–30%, що можна пояснити зменшенням обсягу переданих даних та більш рівномірним навантаженням на систему. Особливо це проявляється при збільшенні інтенсивності потоку, коли класичний підхід починає втрачати ефективність.

Окрім цього, було відзначено підвищення стабільності роботи системи. При використанні попередньої обробки система демонструє більш плавну реакцію на зміну навантаження, що зменшує ризик виникнення затримок або перевантаження окремих компонентів.

Важливо також підкреслити, що досягнуті результати отримані при використанні відносно простих алгоритмів обробки. Це свідчить про те, що навіть базові механізми фільтрації та агрегації можуть суттєво впливати на ефективність роботи системи.

Разом з тим, запропоноване рішення має певні обмеження. Зокрема, використання простих методів виявлення аномалій не дозволяє охопити всі можливі сценарії. Крім того, система розглядається як демонстраційна модель і не враховує всіх особливостей реальних розподілених середовищ.

Попри це, отримані результати підтверджують доцільність використання периферійних обчислень для попередньої обробки поточкових даних. Запропонований підхід дозволяє зменшити навантаження на систему, зберігаючи при цьому необхідний рівень швидкодії.

Таким чином, можна зробити висновок, що реалізована система досягає поставленої мети та демонструє ефективність застосування edge computing у задачах обробки поточкових даних. Отримані результати можуть бути використані як основа для подальшого розвитку більш складних систем у даній області.

3.8 Висновки до третього розділу

У третьому розділі було реалізовано програмно-апаратну систему попередньої обробки поточкових даних та проведено аналіз її роботи в умовах використання периферійних обчислень. Розроблено модулі генерації та прийому даних, які забезпечують формування потоків подій, їх передачу та фіксацію результатів обробки.

Основну увагу приділено реалізації підсистеми попередньої обробки на edge-рівні. Було реалізовано механізми фільтрації, агрегації та виявлення аномалій, які дозволяють скоротити обсяг переданих даних та підвищити їх інформативність. Встановлено, що використання послідовного конвеєра обробки забезпечує узгоджену роботу всіх механізмів і дозволяє ефективно працювати з потоковими даними в режимі реального часу.

У процесі роботи також виконано інтеграцію окремих компонентів у єдину систему. Забезпечено узгоджену взаємодію апаратного edge-вузла, програмного застосунку, модулів обробки, прийому даних та підсистеми метрик. Визначено, що використання асинхронного підходу дозволяє уникнути блокування компонентів і забезпечує стабільну роботу системи при різній інтенсивності потоку.

Для оцінки ефективності системи було проведено серію експериментів із різними сценаріями надходження даних. Проаналізовано поведінку системи при змінному навантаженні, а також у режимах із використанням edge-обробки та без неї. Отримані результати показали, що попередня обробка дозволяє суттєво зменшити кількість переданих повідомлень і знизити навантаження на систему.

Проведене оцінювання продуктивності підтвердило ефективність запропонованого підходу. Встановлено, що використання периферійної обробки дозволяє скоротити обсяг переданих даних у середньому у 2–2,5 рази та зменшити затримку обробки приблизно на 20–30%. Крім того, система демонструє більш стабільну роботу при зростанні інтенсивності потоку.

У результаті дослідження підтверджено доцільність використання edge computing для задач попередньої обробки поточкових даних. Реалізована програмно-апаратна система забезпечує ефективне поєднання простоти реалізації, наочності роботи та достатнього рівня функціональності, що створює основу для подальшого розвитку більш складних рішень у даній області.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було розглянуто підходи до обробки потокових даних та досліджено можливості використання периферійних обчислень у сучасних інформаційних системах. Встановлено, що традиційна централізована обробка не завжди є ефективною у випадках швидкого зростання обсягів даних, підвищених вимог до швидкодії та необхідності зменшення затримок.

У першому розділі було визначено основні особливості потокових даних, вимоги до їх обробки та обмеження традиційних підходів. Розглянуто концепцію edge computing, архітектурні підходи до побудови систем попередньої обробки та сучасні технології роботи з потоками даних. Це дозволило підтвердити доцільність використання попередньої обробки на периферійному рівні.

У другому розділі було розроблено концепцію та структуру програмно-апаратної системи попередньої обробки потокових даних. Запропонована архітектура передбачає використання апаратного edge-вузла на базі ESP32, Arduino-програми для локальної обробки даних та програмного застосунку C# для моделювання, візуалізації й порівняння режимів роботи системи.

У третьому розділі було реалізовано програмно-апаратний прототип системи та проведено перевірку його роботи. Апаратна частина забезпечує імітацію надходження потокових даних за допомогою потенціометра, перемикання режимів роботи за допомогою кнопки та виведення результатів роботи програми мікроконтролера у Serial Monitor. Програмна частина забезпечує моделювання потоків даних, відображення результатів обробки та формування показників ефективності.

У роботі реалізовано механізми фільтрації, агрегації та виявлення аномалій. Вони дозволяють виконувати попередню обробку даних на edge-рівні, зменшувати кількість повідомлень, що передаються далі, та формувати більш інформативні результати для подальшого використання.

Проведені експерименти дозволили оцінити поведінку системи в різних умовах. Було перевірено режими роботи без edge-обробки та з використанням попередньої обробки. Отримані результати показали, що застосування edge-рівня дозволяє зменшити кількість переданих повідомлень і загальний обсяг даних. У середньому обсяг трафіку скорочується приблизно у 2–2,5 рази.

Оцінювання затримок обробки показало, що впровадження edge-рівня не погіршує швидкодію системи. Навпаки, у ряді випадків спостерігається зменшення затримки з 40–45 мс до 25–30 мс, що відповідає покращенню приблизно на 20–30%. Це свідчить про ефективність використання попередньої обробки для зменшення навантаження на систему.

Окрім цього, було відзначено підвищення стабільності роботи системи при зміні інтенсивності потоку. Попередня обробка дозволяє більш рівномірно розподіляти навантаження та зменшує ризик перевантаження окремих компонентів. При цьому реалізована програмно-апаратна система залишається достатньо простою, наочною та придатною для демонстрації основних принципів edge computing.

Разом з тим, слід зазначити, що розроблена система має демонстраційний характер. Використані алгоритми обробки є спрощеними і можуть бути розширені у подальших дослідженнях. Зокрема, перспективним напрямом є використання складніших методів обробки даних, підключення реальних сенсорів та інтеграція з потоковими платформами.

Таким чином, поставлена мета роботи була досягнута. Розроблена програмно-апаратна система підтверджує доцільність використання периферійних обчислень для попередньої обробки поточкових даних та може бути використана як основа для подальших досліджень і розробок у цій області.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Satyanarayanan M. The Emergence of Edge Computing. *Computer*. 2017. Vol. 50, No. 1. P. 30–39.
2. Shi W., Dustdar S. The Promise of Edge Computing. *Computer*. 2016. Vol. 49, No. 5. P. 78–81.
3. Zhou Z., Chen X., Li E., Zeng L., Luo K., Zhang J. Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing. *Proceedings of the IEEE*. 2019. Vol. 107, No. 8. P. 1738–1762.
4. Chiang M., Zhang T. Fog and IoT: An Overview of Research Opportunities. *IEEE Internet of Things Journal*. 2016. Vol. 3, No. 6. P. 854–864.
5. Ray P. P., Dash D., De D. Edge Computing for Internet of Things: A Survey, E-Healthcare Case Study and Future Direction. *Journal of Network and Computer Applications*. 2019. Vol. 140. P. 1–22.
6. Linthicum D. Connecting Fog and Cloud Computing. *IEEE Cloud Computing*. 2017. Vol. 4, No. 2. P. 18–20.
7. Varghese B., Buyya R. Next Generation Cloud Computing: New Trends and Research Directions. *Future Generation Computer Systems*. 2018. Vol. 79. P. 849–861.
8. Roman R., Lopez J., Mambo M. Mobile Edge Computing, Fog et al.: A Survey and Analysis of Security Threats and Challenges. *Future Generation Computer Systems*. 2018. Vol. 78. P. 680–698.
9. Mach P., Becvar Z. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Communications Surveys & Tutorials*. 2017. Vol. 19, No. 3. P. 1628–1656.
10. 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.

11. Akidau T., Chernyak S., Lax R. *Streaming Systems: The What, Where, When, and How of Large-Scale Data Processing*. Sebastopol : O'Reilly Media, 2018. 328 p.

12. Kleppmann M. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. Sebastopol : O'Reilly Media, 2017. 616 p.

13. Fragkoulis M., Carbone P., Kalavri V., Katsifodimos A. A Survey on the Evolution of Stream Processing Systems. *The VLDB Journal*. 2024. Vol. 33, No. 2. P. 507–541.

14. Bifet A., Gavaldà R., Holmes G., Pfahringer B. *Machine Learning for Data Streams: with Practical Examples in MOA*. Cambridge : MIT Press, 2018. 288 p.

15. Karimov J., Rabl T., Katsifodimos A., Samarev R., Heiskanen H., Markl V. Benchmarking Distributed Stream Data Processing Systems. *Proceedings of the IEEE 34th International Conference on Data Engineering (ICDE)*. 2018. P. 1519–1530.

16. Verwiebe J., Grulich P. M., Traub J., Markl V. Survey of Window Types for Aggregation in Stream Processing Systems. *The VLDB Journal*. 2023. Vol. 32, No. 5. P. 985–1011.

17. Dias de Assunção M., da Silva Veith A., Buyya R. Distributed Data Stream Processing and Edge Computing: A Survey on Resource Elasticity and Future Directions. *Journal of Network and Computer Applications*. 2018. Vol. 103. P. 1–17.

18. Vikash, Mishra L., Varma S. Performance Evaluation of Real-Time Stream Processing Systems for Internet of Things Applications. *Future Generation Computer Systems*. 2020. Vol. 113. P. 207–217.

19. Noghabi S. A. et al. Samza: Stateful Scalable Stream Processing at LinkedIn. *Proceedings of the VLDB Endowment*. 2017. Vol. 10, No. 12. P. 1634–1645.

20. Traub J., Grulich P. M., Rodríguez Cuéllar A., Breß S., Katsifodimos A., Rabl T., Markl V. Efficient Window Aggregation with General Stream Slicing. *Proceedings of the 22nd International Conference on Extending Database Technology (EDBT)*. 2019. P. 97–108.

21. Richards M. *Software Architecture Patterns*. 2nd ed. Sebastopol : O'Reilly Media, 2022. 72 p.
22. Richards M., Ford N. *Fundamentals of Software Architecture: An Engineering Approach*. Sebastopol : O'Reilly Media, 2020. 432 p.
23. Van Steen M., Tanenbaum A. S. *Distributed Systems*. 3rd ed. CreateSpace Independent Publishing Platform, 2017. 582 p.
24. Stopford B. *Designing Event-Driven Systems: Concepts and Patterns for Streaming Services with Apache Kafka*. Sebastopol : O'Reilly Media, 2018. 171 p.
25. Newman S. *Building Microservices: Designing Fine-Grained Systems*. 2nd ed. Sebastopol : O'Reilly Media, 2021. 612 p.
26. Feick M., Kleer N., Kohn M. Fundamentals of Real-Time Data Processing Architectures Lambda and Kappa. *SKILL 2018 – Studierendenkonferenz Informatik*. Bonn : Gesellschaft für Informatik, 2018. P. 55–66.
27. Giebler C., Stach C., Schwarz H., Mitschang B. BRAID: A Hybrid Processing Architecture for Big Data. *Proceedings of the 7th International Conference on Data Science, Technology and Applications (DATA 2018)*. 2018. P. 294–301.
28. Hamdan S., Ayyash M., Almajali S. Edge-Computing Architectures for Internet of Things Applications: A Survey. *Sensors*. 2020. Vol. 20, No. 22. Article 6441.
29. Bellemare A. *Building Event-Driven Microservices: Leveraging Organizational Data at Scale*. Sebastopol : O'Reilly Media, 2020. 321 p.
30. Burns B. *Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services*. Sebastopol : O'Reilly Media, 2018. 166 p.
31. Cheraghlou M. N., Khadem-Zadeh A., Haghparast M. A Survey of Fault Tolerance Architecture in Cloud Computing. *Journal of Network and Computer Applications*. 2016. Vol. 61. P. 81–92.
32. Mao Y., You C., Zhang J., Huang K., Letaief K. B. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys & Tutorials*. 2017. Vol. 19, No. 4. P. 2322–2358.

33. Shapira G., Palino T., Sivaram R., Petty K. *Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale*. 2nd ed. Sebastopol : O'Reilly Media, 2021. 485 p.

34. Apache Software Foundation. *Apache Kafka Documentation*. 2023. URL: <https://kafka.apache.org/documentation/> (дата звернення: 16.04.2026).

35. Roy G. M. *RabbitMQ in Depth*. Shelter Island : Manning Publications, 2017. 264 p.

36. Apache Software Foundation. *Apache Pulsar Documentation*. 2023. URL: <https://pulsar.apache.org/docs/> (дата звернення: 16.04.2026).

37. Hueske F., Kalavri V. *Stream Processing with Apache Flink: Fundamentals, Implementation, and Operation of Streaming Applications*. Sebastopol : O'Reilly Media, 2019. 308 p.

38. Apache Software Foundation. *Apache Flink Documentation*. 2023. URL: <https://nightlies.apache.org/flink/flink-docs-release-1.18/> (дата звернення: 16.04.2026).

39. Zaharia M. et al. Apache Spark: A Unified Engine for Big Data Processing. *Communications of the ACM*. 2016. Vol. 59, No. 11. P. 56–65.

40. Apache Software Foundation. *Apache Spark Streaming Programming Guide*. 2022. URL: <https://spark.apache.org/docs/latest/streaming-programming-guide.html> (дата звернення: 16.04.2026).

41. Apache Software Foundation. *Apache Storm Documentation*. 2021. URL: <https://storm.apache.org/releases/current/index.html> (дата звернення: 16.04.2026).

42. Apache Software Foundation. *Apache Beam Documentation*. 2023. URL: <https://beam.apache.org/documentation/> (дата звернення: 16.04.2026).

43. Apache Software Foundation. *Apache Beam Programming Guide*. 2023. URL: <https://beam.apache.org/documentation/programming-guide/> (дата звернення: 16.04.2026).

44. Premsankar G., Di Francesco M., Taleb T. Edge Computing for the Internet of Things: A Case Study. *IEEE Internet of Things Journal*. 2018. Vol. 5, No. 2. P. 1275–1284.

45. Nickoloff J., Kuenzli S. *Docker in Action*. 2nd ed. Shelter Island : Manning Publications, 2019. 336 p.

46. Burns B., Beda J., Hightower K. *Kubernetes: Up and Running: Dive into the Future of Infrastructure*. 2nd ed. Sebastopol : O'Reilly Media, 2019. 278 p.

47. Jensen S. K., Pedersen T. B., Thomsen C. Time Series Management Systems: A Survey. *IEEE Transactions on Knowledge and Data Engineering*. 2017. Vol. 29, No. 11. P. 2581–2600.

48. Prometheus. *Visualizing metrics using Grafana*. Prometheus Documentation. URL: https://prometheus.io/docs/tutorials/visualizing_metrics_using_grafana/ (дата звернення: 16.04.2026).

49. Dastjerdi A. V., Buyya R. Fog Computing: Helping the Internet of Things Realize Its Potential. *Computer*. 2016. Vol. 49, No. 8. P. 112–116.

50. Kleppmann M. *Making Sense of Stream Processing: The Philosophy Behind Apache Kafka and Scalable Stream Data Platforms*. Sebastopol : O'Reilly Media, 2016. 181 p.

51. Isah H., Abughofa T., Mahfuz S., Ajerla D., Zulkernine F., Khan S. A Survey of Distributed Data Stream Processing Frameworks. *IEEE Access*. 2019. Vol. 7. P. 154300–154316.

52. Apache Software Foundation. *Structured Streaming Programming Guide*. Apache Spark Documentation. 2023. URL: <https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html> (дата звернення: 16.04.2026).

53. Dobbelaere P., Esmaili K. S. Kafka versus RabbitMQ: A Comparative Study of Two Industry Reference Publish/Subscribe Implementations. *Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems*. 2017. P. 227–238.

54. 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.

55. Nygard M. *Release It!: Design and Deploy Production-Ready Software*. 2nd ed. Raleigh : Pragmatic Bookshelf, 2018. 376 p.

56. Hong C.-H., Varghese B. Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms. *ACM Computing Surveys*. 2019. Vol. 52, No. 5. Article 97.

57. Majors C., Fong-Jones L., Miranda G. *Observability Engineering: Achieving Production Excellence*. Sebastopol : O'Reilly Media, 2022. 318 p.

58. Microsoft. *Asynchronous Programming Patterns*. Microsoft Docs. 2023. URL: <https://learn.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/> (дата звернення: 17.04.2026).

59. Microsoft. *C# Documentation*. Microsoft Docs. 2023. URL: <https://learn.microsoft.com/en-us/dotnet/csharp/> (дата звернення: 17.04.2026).

60. Microsoft. *.NET Framework Overview*. Microsoft Docs. 2022. URL: <https://learn.microsoft.com/en-us/dotnet/framework/get-started/overview> (дата звернення: 17.04.2026).

61. Microsoft. *WPF Overview*. Microsoft Docs. 2023. URL: <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/> (дата звернення: 17.04.2026).

62. Microsoft. *Model-View-ViewModel (MVVM)*. Microsoft Learn. 2024. URL: <https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm> (дата звернення: 17.04.2026).

63. Toub S. *Async Programming in C#*. Microsoft Docs. 2021. URL: <https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/> (дата звернення: 17.04.2026).

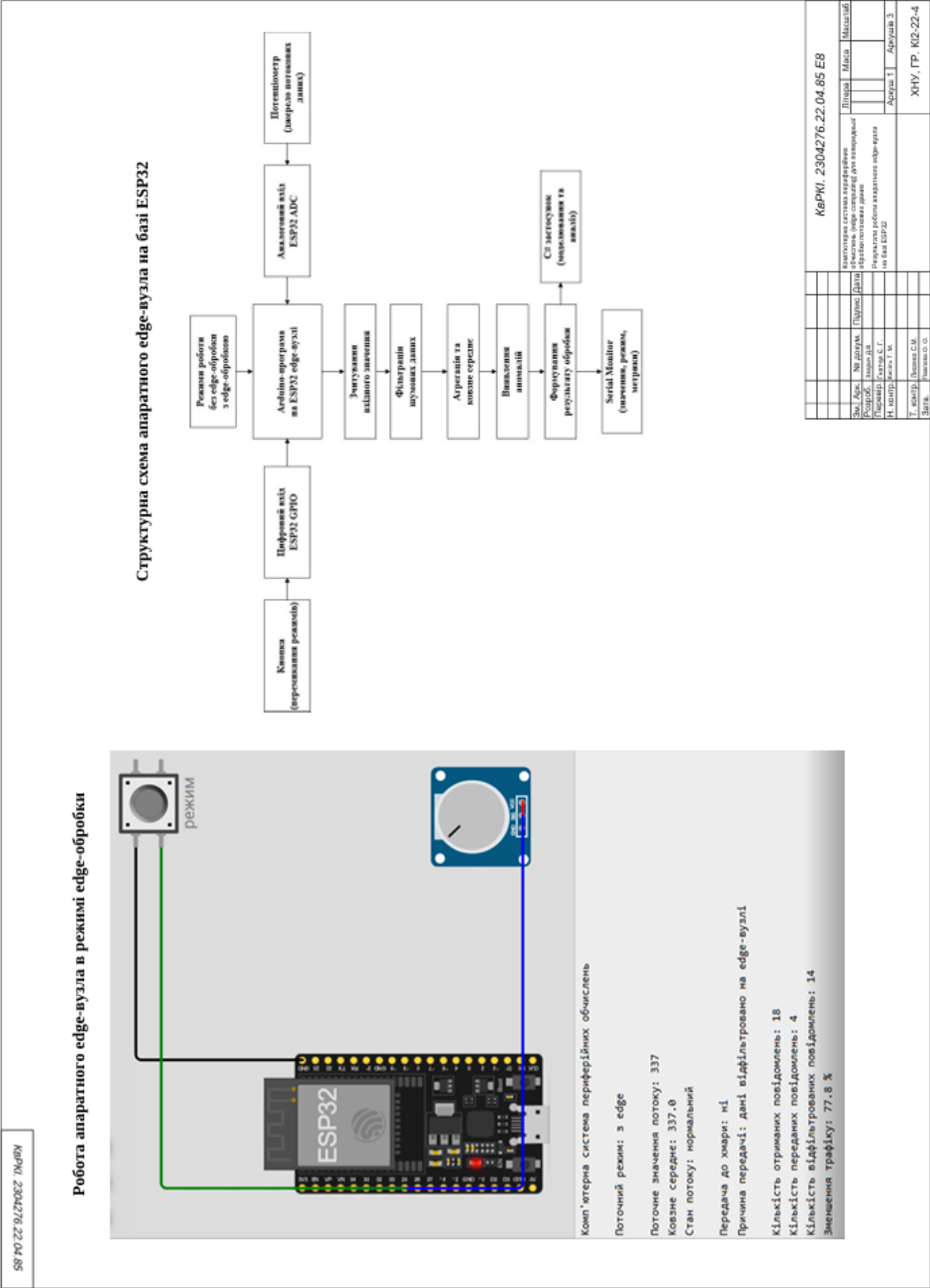
Копія креслення «Результати роботи програмного застосунку»



ДОДАТОК Б

(обов'язковий)

Копія креслення «Результати роботи апаратного edge-вузла на базі ESP32»



Копія креслення «Структурні схеми програмно-апаратної системи»



Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Данило ЗВАРИЧ

Співавтор:

Назва: Комп'ютерна система периферійних обчислень (edge computing) для попередньої обробки потокових даних

Експерт: Єлизавета ГНАТЧУК

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1: 6.64%

Коефіцієнт подібності 2: 1.48%

Мікропробіли: 3

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2026-06-06 11:44:59.0

Після аналізу Звіту подібності констатую наступне:

☒ Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.

☐ Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.

☐ Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедур. Таким чином робота не приймається.

Обґрунтування:

2026-06-06

Дата



Доцент Андрій Нічепорук

експерт

Anti-Plagiarism (<http://ap.km.ua>) v-15.701

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. Помилок в документах: 4%

ID: 273897 Назва: БКР Комп'ютерна система периферійних обчислень (edge computing) для попередньої обробки поточкових даних Додано в БД: 2026-06-06 Автора: Данило ЗВАРИЧ Керівники: Єлизавета ГНАТЧУК Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	107438	995	3299 (3%)	45 (5%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Зварич Данило Віталійович

Тема: Комп'ютерна система периферійних обчислень (edge computing) для попередньої обробки потокових даних

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 Кількість сторінок записки 74

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є розробка та дослідження комп'ютерної системи периферійних обчислень для попередньої обробки потокових даних.
2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.
3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі кваліфікаційної роботи проведено дослідження предметної області периферійних обчислень та потокових даних, розглянуто архітектурні підходи до побудови систем попередньої обробки, сучасні технології потокової обробки та існуючі рішення, визначено їх переваги й обмеження, а також сформульовано мету та задачі дослідження. У другому розділі виконано проектування та моделювання програмно-апаратної системи периферійних обчислень, зокрема визначено загальну концепцію побудови системи, спроектовано архітектуру апаратного edge-вузла на базі ESP32, описано сценарії надходження потокових даних, механізми фільтрації, агрегації та трансформації даних, обґрунтовано вибір технологічного стеку і структуру обміну повідомленнями між компонентами. У третьому розділі здійснено програмно-апаратну реалізацію системи: розроблено модулі збору та прийому потокових даних, реалізовано попередню обробку на edge-рівні, механізми фільтрації, агрегації та виявлення аномалій, інтегровано апаратний вузол і програмний застосунок, проведено експерименти з потоковими даними, досліджено продуктивність, затримки обробки та

ефективність запропонованого рішення. У роботі використано сучасні підходи edge computing, принципи роботи IoT-систем, асинхронну обробку даних, мікроконтролерну платформу ESP32 та засоби розробки програмних застосунків на C#.

4. Позитивні сторони роботи: практична цінність роботи, використання програмно-апаратного підходу, реалізація апаратного edge-вузла на базі ESP32 та наочне порівняння режимів обробки потокових даних.

5. Негативні сторони роботи: недостатня увага перевірці системи з реальними сенсорними даними.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на належному науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: добре (75/С)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Гейлек Наталія Сергіївна, доцент кафедри КВ

"11" серпня 2026 р.

 (підпис)

Зав. кафедри КІПС
д-р. філософії Ользі ПАВЛОВІЙ

Данило ЗВАРИЧ

ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи КІ2-22-4

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів академічної відповідальності, ознайомлений (а). Про використання спеціалізованих програмних засобів (СПЗ) StrikePlagiarism та Anti-Plagiarism для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а). Надаю університету право на передачу моєї роботи для обробки та збереження в базах даних СПЗ і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються СПЗ.

Також надаю свою згоду на обробку й збереження університетом моєї роботи в Інституційному репозитарії Хмельницького національного університету.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

1 травня 2026 року



РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Комп'ютерна система периферійних обчислень (edge computing) для попередньої обробки потокових даних

Автор Данило ЗВАРИЧ

Освітня програма Комп'ютерна інженерія та програмування

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 Комп'ютерна інженерія

Науковий керівник: д.т.н., проф. Єлизавета ГНАТЧУК

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	відповідає
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

- 1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
- 2) окремі виявлені збіги є загальноживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;
- 3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.
- 4) значна частина знайденого плагіату відноситься до списку використаних джерел

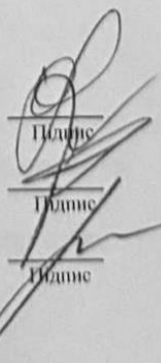
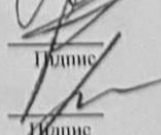
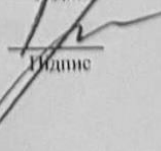
Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 6,64%; та системою Anti-Plagiarism складає 1%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

01.06.2026

Завідувач кафедри

Гарант освітньої програми

Керівник кваліфікаційної роботи


Підпис

Підпис

Підпис

Ольга ПАВЛОВА
Ім'я, ПРІЗВИЩЕ

Андрій НІЧЕПОРУК
Ім'я, ПРІЗВИЩЕ

Єлизавета ГНАТЧУК
Ім'я, ПРІЗВИЩЕ