https://doi.org/10.31649/1997-9266-2026-186-3-99-109

УДК 004.62:004.8:004.65

О. В. Бойко[1]
М. В. Добролюбова[1, 2]

СИСТЕМА САМОКАЛІБРУВАННЯ СЕНСОРІВ РОЗУМНОГО БУДИНКУ НА ОСНОВІ РОБАСТНОЇ РЕГРЕСІЇ ТА ОНЛАЙН-ДЕТЕКЦІЇ АНОМАЛІЙ

[1]Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»;
[2]Київський національний економічний університет імені Вадима Гетьмана

Розглянуто програмну платформу самокалібрування сенсорів розумного будинку, реалізовану на Python, як зовнішній сервіс для локальних систем Інтернету речей (IoT). Актуальність роботи зумовлена тим, що дрейф і систематична похибка недорогих температурних та вологісних сенсорів безпосередньо впливають на якість моніторингу мікроклімату, коректність автоматизованих сценаріїв керування опаленням і вентиляцією, а також на енергетичну ефективність житла. На рівні програмної реалізації відтворено повний конвеєр обробки даних: надходження показів через REST API, MQTT або симулятор, збереження в базі даних PostgreSQL, періодичне перенавчання калібрувальної моделі, застосування корекції до нових вимірювань, онлайн-виявлення аномалій та подальшу візуалізацію через веб-дашборд і Grafana. Методична новизна рішення полягає у поєднанні часової синхронізації пар «сирий сенсор – еталон», робастної афінної корекції за допомогою HuberRegressor, правила прийняття моделі за критерієм зменшення середньої абсолютної похибки (mean absolute error, MAE) не менше ніж на 2 % та онлайн-детекції аномалій на основі медіанного абсолютного відхилення (median absolute deviation, MAD) з резервним переходом до z-оцінки. Показано, що алгоритм не лише навчає корекційну модель, а і може безпечно відмовлятися від її застосування, якщо якість моделі недостатня або даних для навчання замало. Експеримент на вбудованому симуляторі, який моделює чотири кімнати, 20 сенсорів і 4800 вимірювань, засвідчив виражене зменшення похибки для температурних каналів: середня MAE після початкового калібрування зменшилася з 1,56 до 0,31 °C. Для каналів вологості ефект виявився помірнішим: середня похибка знизилася з 1,44 до 1,34 %RH. У завершальному онлайновому вікні після серії перенавчань середнє зменшення MAE для всіх парних каналів становило 30,4 %, при цьому для температурних сенсорів воно досягло 42,3 %, а для вологісних — 14,5 %. Додаткова перевірка на архівній SQLite-базі з проєкту підтвердила сильний ефект для температурного датчика у вітальні (1,789 → 0,162) та помірний ефект для вологісного каналу (0,611 → 0,588). Практична цінність запропонованого підходу полягає в тому, що його можна інтегрувати у локальну інфраструктуру розумного будинку без модифікації прошивки кінцевих пристроїв, підвищуючи достовірність телеметрії, зменшуючи кількість хибних спрацювань та створюючи передумови для економії енергії в системах домашньої автоматизації. Окрім технічного ефекту, запропоноване рішення має виражене прикладне соціально-економічне значення, оскільки дає змогу підвищити якість роботи дешевих сенсорів програмними засобами без переходу на дорожче обладнання. Це суттєво полегшує повсякденне користування бюджетними системами розумного дому, робить такі системи доступнішими для ширшого кола споживачів і в перспективі сприяє зменшенню їх загальної вартості як на етапі впровадження, так і під час подальшої експлуатації.

Ключові слова: розумний будинок; самокалібрування сенсорів; робастна регресія; HuberRegressor; середня абсолютна похибка; медіанне абсолютне відхилення; виявлення аномалій; MQTT; FastAPI; PostgreSQL; Grafana, програмування, база даних.

Вступ

Надійність системи розумного будинку безпосередньо залежить від достовірності сенсорних вимірювань. Помилка в температурі, вологості або концентрації CO_2 зумовлює не лише неточне відображення стану середовища, а й помилкові автоматизовані рішення: передчасне вимкнення опалення, некоректний запуск вентиляції або хибні тривоги. Для низьковартісних IoT-пристроїв

ця проблема є особливо гострою, оскільки дрейф, старіння чутливих елементів, похибки монтажу та нестабільність умов експлуатації накопичуються швидше ніж у професійних вимірювальних системах. Саме тому сучасні дослідження з машинного навчання для калібрування сенсорів фокусуються на адаптивних моделях, які можуть компенсувати похибку без лабораторного втручання [1]—[3].

Окремим напрямом є виявлення аномалій у потоках IoT-даних. Огляд AI-підходів до anomaly detection у сенсорних мережах показує, що для практичних систем важливими є не лише точність, а і невелика обчислювальна складність, локальне розгортання та інтерпретованість результату [4]. У такому контексті прості робастні статистичні процедури, зокрема MAD, залишаються конкурентоспроможними, коли необхідно мати легкий модуль онлайн-оцінювання без великих обчислювальних витрат [5].

Розроблений програмний додаток поєднує практичну інженерію з науково осмисленою логікою обробки даних. На відміну від багатьох демонстраційних проєктів класу smart home, тут реалізовано не лише веб-інтерфейс і збереження даних, та окремий калібрувальний шар, який працює на історичних парах «сирий вимір – еталон», оцінює дрейф, приймає або відхиляє модель і повертає скориговані значення у подальший конвеєр обробки. Таке рішення потенційно придатне для інтеграції з локальними MQTT-екосистемами та системами домашньої автоматизації без зміни прошивки кінцевих пристроїв [6], [7].

Отже, *метою статті* є підвищення достовірності моніторингу показників мікроклімату та зниження експлуатаційних і енергетичних витрат у системах розумного будинку шляхом розроблення, формалізації та експериментальної оцінки програмної системи самокалібрування сенсорів. Для досягнення цієї мети необхідно: відтворити архітектуру програмного комплексу; формалізувати алгоритм навчання та застосування калібрувальної моделі; дослідити механізм виявлення аномалій; оцінити ефект корекції в симульованому та архівному сценаріях.

Архітектура програмної системи

Архітектура проєкту побудована за сервісним принципом. Файл main.py ініціалізує FastAPI-застосунок, налаштовує життєвий цикл, запускає симулятор і фоновий цикл перенавчання, а також відкриває REST та WebSocket-інтерфейси. На нижчому рівні repository.py та database.py реалізують доступ до бази даних (БД) PostgreSQL і створюють таблиці sensors, readings та calibration_models разом із представленнями для дашбордів. Бізнес-логіка винесена у такі сервіси:

– IngestService — приймає нові покази;

– CalibrationService — навчає та застосовує моделі;

– SimulatorService — генерує синтетичні сценарії;

– MQTTIngestService — забезпечує інтеграцію з брокером MQTT;

– ConnectionManager — транслює події в реальному часі до веб-клієнтів.

Узагальнену схему показано на рис. 1.

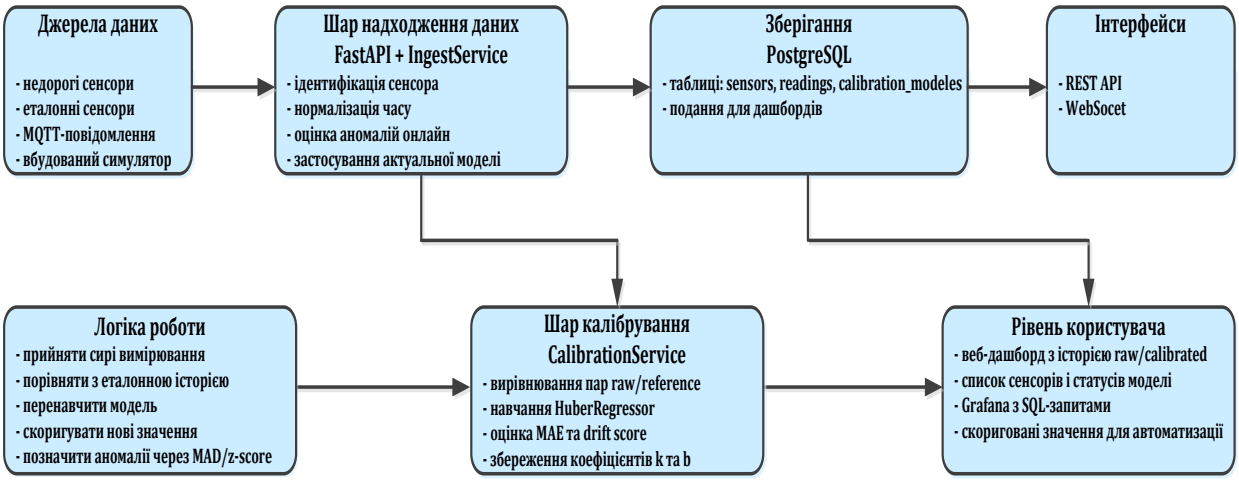

Рис. 1. Архітектура платформи самокалібрування, реконструйована за вихідним кодом

Основні модулі системи та їх функції подані в табл. 1.

Таблиця 1

Основні модулі системи та їх функції

Компонент	Основна роль	Вхід / вихід
app/main.py	керування життєвим циклом, REST, WebSocket, запуск фонових задач	HTTP, WebSocket, періодичне перенавчання
services/ingest_service.py	нормалізація пакета, виклик apply(), оцінка аномалій	ReadingCreate → readings
services/calibration_service.py	вирівнювання пар, навчання HuberRegressor [8], оцінка drift	історія вимірювань → slope, intercept, status
services/simulator_service.py	генерація еталонних і дрейфуючих даних, ін'єкція збурень	синтетичні reading-події
services/mqtt_service.py	розбір MQTT-повідомлень та їх перетворення у ReadingCreate	topic/payload → ingest
repository.py + database.py	зберігання сенсорів, вимірювань, моделей, SQL-подання	PostgreSQL CRUD + dashboard views

Інженерна перевага такої декомпозиції полягає у відокремленні калібрувальної логіки від транспортного та візуалізаційного шарів. Дані можуть надходити через REST, MQTT або симулятор, але після цього проходять однаковий маршрут: ідентифікація сенсора, пошук актуальної моделі, корекція значення, аномальний контроль, запис у БД і розсилка оновлень. Це дає змогу замінювати джерело даних без модифікації самого алгоритму самокалібрування. До того ж веб-клієнт і Grafana працюють з уже підготовленими таблицями та поданнями, а не з логікою моделі напряму, що спрощує розгортання та тестування [9], [10].

Код також чітко демонструє орієнтацію на локальне розгортання. За замовчуванням застосунок використовує MQTT-теми виду smart-home/+/+, сумісні з легким publish/subscribe обміном, який стандартизовано в MQTT 5.0 [6]. Офіційна документація Home Assistant підтверджує можливість локальної MQTT-інтеграції та використання протоколу версії 5 у домашній автоматизації [7]. Для аналітичної візуалізації вибрано Grafana з вбудованим PostgreSQL data source, що не потребує встановлення окремого плагіна та напряму підтримує SQL-дашборди [10].

Метод самокалібрування та онлайн-детекції аномалій

Алгоритм самокалібрування реалізовано в класі *CalibrationService*. Для кожного нееталонного сенсора, що має *reference_sensor_id*, сервіс отримує історію вимірювань поточного каналу та відповідного еталона. Далі будується список вирівняних пар за часовими мітками. У коді використано лінійне проходження з рухомим покажчиком *ref_index* і локальним пошуком серед трьох кандидатів (поточний еталонний запис, попередній і наступний). Такий підхід має фактично лінійну складність щодо довжини відсортованих рядів і є значно дешевшим за повний попарний пошук. Пара приймається лише тоді, коли часовий розрив не перевищує 600 с. Для навчання використовується не більше 300 останніх узгоджених пар, а мінімально допустимий обсяг становить 12 точок.

Після вирівнювання даних формується одновимірна задача регресії, де x — це *raw_value* проблемного сенсора, а y — *raw_value* еталона. Кориговане значення описується афінною моделлю

$$\hat{y} = kx + b, \tag{1}$$

де k — коефіцієнт масштабу, а b — зсув. Для оцінювання k та b застосовано HuberRegressor [8], тобто робастну регресію, що поєднує квадратичну втрату на малих похибках і лінійну — на великих відхиленнях. Такий вибір добре узгоджується з практикою сенсорних систем, у яких окремі аномальні сплески або розсинхронізації не повинні зруйнувати всю оцінку моделі [11]. Сам код не вводить додаткових предикторів середовища, тому метод варто розглядати як компактну та інтерпретовану однофакторну корекцію.

Якість моделі оцінюється через середню абсолютну похибку між «сирими» та еталонними даними до корекції, а також між прогнозом моделі та еталоном після корекції:

$$MAE = (1/n)\sum_{i=1}^{n}\left|x_i - y_i\right|, \tag{2}$$

де n — кількість спостережень (даних), x_i — істинне (реальне) значення для i-го прикладу, y_i — передбачене моделлю значення для i-го прикладу, $\left|x_i - y_i\right|$ — абсолютна похибка для одного прикладу (без знаку).

У коді *improvement = MAE_before – MAE_after*, а модель приймається лише тоді, коли кандидат зменшує похибку не менше ніж на 2 %, тобто *MAE_after* < 0,98 * *MAE_before*. Якщо умова не виконується, сервіс зберігає нейтральні параметри $k = 1$ і $b = 0$. Додатково *drift_score* прирівнюється до *MAE_before* і порівнюється з порогом *drift_threshold* (за замовчуванням 0,7). Звідси формується статус: *calibrated* — якщо дрейф істотний і корекцію прийнято; *attention* — якщо дрейф істотний, але модель не дала покращення; *stable* — якщо рівень дрейфу не перевищив поріг. Така логіка важлива, оскільки дозволяє відокремити саме факт виявлення проблеми від факту успішної математичної компенсації.

Застосування моделі до нових даних відбувається у методі *apply()*. Якщо для сенсора ще немає моделі, значення пропускається без змін. Якщо останній статус належить до *attention* або *insufficient_data*, система також повертає вихідний *raw_value*, оскільки корекція вважається ненадійною. В усіх інших випадках обчислюється *calibrated_value = k * raw_value + b*, а *residual* фіксує відхилення між скоригованим і вихідним показом. Саме це рішення є важливим для експлуатації: система не лише шукає модель, а може безпечно відмовитися від її застосування, якщо довіра до неї недостатня.

Виявлення аномалій реалізоване у *IngestService*. Для кожного нового вимірювання сервіс бере до 60 останніх *calibrated_value* того самого сенсора, обчислює медіану та *median absolute deviation*, а далі формує робастний *score*

$$z\_MAD = \left| 0,6745 \cdot \left(x - med\left(x \right) / MAD \right) \right|, \tag{3}$$

що узгоджується з рекомендаціями щодо робастної обробки викидів [5]. Якщо MAD наближається до нуля, код переходить до резервної схеми на основі стандартного відхилення. Поріг *anomaly_threshold* для спрацювання за замовчуванням дорівнює 5,0. У такий спосіб у системі поєднано дві різні ролі: калібрування бореться з систематичною похибкою, а anomaly detection — з короткочасними нетиповими сплесками. Повний маршрут обробки показано на рис. 2.

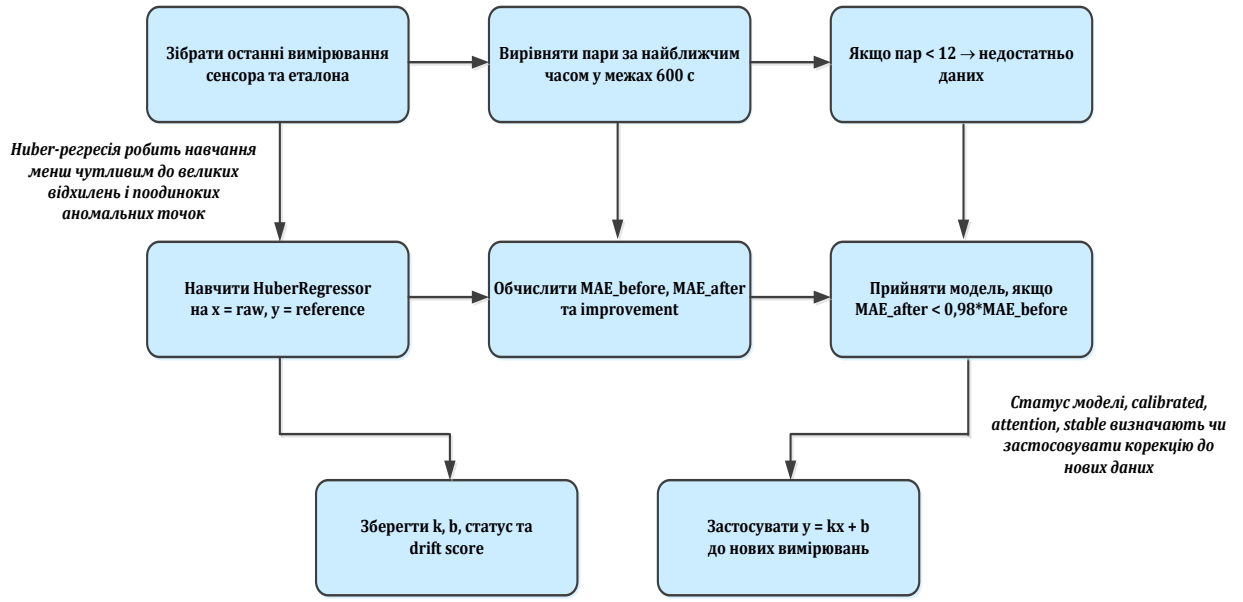

Рис. 2. Послідовність дій алгоритму самокалібрування та прийняття моделі

Ключові параметри алгоритму та їх інтерпретація подані в табл. 2.

Таблиця 2

Ключові параметри алгоритму та їх інтерпретація

Параметр	Значення	Призначення
min_pairs	12	мінімальна кількість вирівняних пар для навчання
match_tolerance_seconds	600 с	максимальний часовий розрив між raw і reference
max_pairs	300	обмеження обсягу історії для навчання
критерій прийняття	MAE_after < 0.98 * MAE_before	фільтр випадків без суттєвого покращення
drift_threshold	0.7	поріг для статусів calibrated / attention / stable

Продовження табл. 2

Параметр	Значення	Призначення
anomaly window	60 точок	обсяг історії для MAD-знаходження викидів
anomaly_threshold	5,0	поріг для позначення аномалії
retrain interval	60 с	інтервал періодичного перенавчання з config.py
simulator interval	5 с	частота надходження нових симульованих показів

Реалізація прототипу та модель даних

На рівні зберігання даних реалізація є достатньо прозорою для подальшої аналітики. Таблиця *sensors* містить метадані каналів, зокрема тип, локацію, одиницю вимірювання, ознаку еталонності та пороги *drift/anomaly*. Таблиця *readings* зберігає часову мітку, *raw_value*, *calibrated_value*, *residual*, джерело події та прапорець аномалії. Таблиця *calibration_models* акумулює всі історичні версії моделі з коефіцієнтами, показниками похибки, *drift_score* і JSON-деталями навчання. Фрагменти коду зі створеними таблицями показані на рис. 3.

```
SCHEMA_STATEMENTS = [
 """
 CREATE TABLE IF NOT EXISTS sensors (
     id BIGSERIAL PRIMARY KEY,
     name TEXT NOT NULL UNIQUE,
     sensor type TEXT NOT NULL,
     location TEXT NOT NULL DEFAULT '',
     unit TEXT NOT NULL DEFAULT '',
     protocol TEXT NOT NULL DEFAULT 'mqtt',
     topic TEXT,
     is reference BOOLEAN NOT NULL DEFAULT FALSE,
     reference_sensor_id BIGINT REFERENCES sensors(id) ON DELETE SET NULL,
     drift_threshold DOUBLE PRECISION NOT NULL DEFAULT 0.7,
     anomaly threshold DOUBLE PRECISION NOT NULL DEFAULT 5.0,
     created_at TIMESTAMPTZ NOT NULL
 )
 """,
 """
 CREATE TABLE IF NOT EXISTS readings (
     id BIGSERIAL PRIMARY KEY,
     sensor id BIGINT NOT NULL REFERENCES sensors(id) ON DELETE CASCADE,
     ts TIMESTAMPTZ NOT NULL,
     raw value DOUBLE PRECISION NOT NULL,
     calibrated_value DOUBLE PRECISION NOT NULL,
     residual DOUBLE PRECISION,
     is anomaly BOOLEAN NOT NULL DEFAULT FALSE,
     source TEXT NOT NULL DEFAULT 'api'
 )
 """,
 """
 CREATE INDEX IF NOT EXISTS idx readings sensor ts
 ON readings(sensor_id, ts DESC)
 """,
 """
 CREATE TABLE IF NOT EXISTS calibration_models (
     id BIGSERIAL PRIMARY KEY,
```

```
with db_cursor() as cursor:
 cursor.execute(
     """
     INSERT INTO sensors (
         name,
         sensor_type,
         location,
         unit,
         protocol,
         topic,
         is_reference,
         reference_sensor_id,
         drift threshold,
         anomaly_threshold,
         created at
     )
     VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
     RETURNING id
     """,
     (
         payload.name,
         payload.sensor type,
         payload.location,
         payload.unit,
         payload.protocol,
         payload.topic,
         payload.is reference,
         payload.reference_sensor_id,
         payload.drift_threshold,
         payload.anomaly threshold,
         created_at,
     ),
 )
 sensor id = int(cursor.fetchone()["id"])
```

Рис. 3. Структура таблиць

Окремо створено SQL-подання *readings_enriched*, *sensor_dashboard_latest* та *dashboard_summary*, які відразу пристосовані до побудови веб-сторінки та Grafana-панелей без дублювання обчислень у фронтенді.

Публічний API сервісу складається з невеликого, але достатнього набору кінцевих точок: /api/readings — для приймання нового показу; /api/calibration/run — для ручного запуску перенавчання; /api/models — для перегляду останніх моделей; /api/sensors та /api/sensors/{sensor_id}/history — для дашбордів; /ws — для оновлення клієнтів у реальному часі (рис. 4).

```
@app.get("/api/sensors")
async def get sensors(request: Request) -> list[dict[str, Any]]:
    return request.app.state.repository.latest_sensor_snapshot()


@app.post("/api/sensors")
async def create sensor(request: Request, payload: SensorCreate) -> dict[str, Any]:
    sensor = request.app.state.repository.create_sensor(payload)
    await request.app.state.websocket_manager.broadcast(
        {"type": "sensor created", "sensor": sensor}
    )
    return sensor


@app.get("/api/sensors/{sensor id}/history")
async def sensor_history(
    request: Request,
    sensor_id: int,
    hours: int = 24,
    limit: int = 300,
) -> dict[str, Any]:
    sensor = request.app.state.repository.get sensor(sensor id)
    if not sensor:
        raise HTTPException(status code=404, detail="Sensor not found")

    readings = request.app.state.repository.get_readings(
        sensor id,
        hours=hours,
        limit=limit,
    )
    return {"sensor": sensor, "readings": readings}
```

```
@app.post("/api/sensors")
async def create_sensor(request: Request, payload: SensorCreate) -> dict[str, Any]:
    sensor = request.app.state.repository.create_sensor(payload)
    await request.app.state.websocket_manager.broadcast(
        {"type": "sensor_created", "sensor": sensor}
    )
    return sensor


@app.get("/api/sensors/{sensor_id}/history")
async def sensor history(
    request: Request,
    sensor id: int,
    hours: int = 24,
    limit: int = 300,
) -> dict[str, Any]:
    sensor = request.app.state.repository.get_sensor(sensor_id)
    if not sensor:
        raise HTTPException(status_code=404, detail="Sensor not found")

    readings = request.app.state.repository.get readings(
        sensor_id,
        hours=hours,
        limit=limit,
    )
    return {"sensor": sensor, "readings": readings}
```

Рис. 4. Набір кінцевих точок

Використання FastAPI тут є технічно вдалим, оскільки фреймворк орієнтований на швидке створення API та має вбудовану підтримку WebSocket-сценаріїв [9]. У результаті калібрування стає не абстрактним офлайн-розрахунком, а частиною живого сервісу, який постійно реагує на появу нових даних.

Важливо підкреслити, що запропонована схема є зовнішньою щодо кінцевих пристроїв. Сенсори не потребують вбудованої логіки самокалібрування і не модифікуються на рівні firmware. Це особливо корисно для бюджетних смарт-хабів і домашніх екосистем, де доступ до низькорівневого ПЗ обмежений. Достатньо, щоб вимірювання надходили в локальний сервіс через MQTT або API, а вже далі система сформує corrected stream, придатний для автоматизації, журналювання та аналітики. Саме така відокремленість і робить розглянутий код як архітектурний патерн для практичних smart-home впроваджень.

Експериментальна методика

Основний експеримент побудовано безпосередньо на вбудованому симуляторі з коду. Він описує чотири кімнати — вітальню, спальню, кухню та кабінет. Для кожної кімнати створюється п’ять каналів: еталонна температура, недорогий температурний сенсор, еталонна вологість, недорогий сенсор вологості та сенсор CO_2. Отже, повна конфігурація містить 20 сенсорів, з яких 8 є парами «низьковартісний – еталон» для задачі калібрування. Базові сигнали генеруються як хвильові добові профілі з випадковим шумом, а недорогі канали додатково отримують систематичний scale/offset зсув і поступовий дрейф. Для температури код фактично реалізує вираз *temp_lowcost = temp_reference * 1.018 + 0.28 + drift + noise*, а для вологості *humidity_lowcost = humidity_reference * 0.962 - 1.4 + drift + noise*.

Під час warm-up етапу згенеровано 180 пакетів вимірювань з кроком 20 с, що імітує накопичення історії перед першим запуском перенавчання. Після цього виконано первинний batch-калібрувальний запуск, а далі змодельовано 60 онлайнових кроків з періодом 5 с і перенавчанням кожні 12 кроків, тобто приблизно раз на хвилину, як у конфігурації застосунку. Для оцінки якості використано MAE і RMSE між недорогим сенсором і відповідним еталоном. Усі підсумкові online-метрики обчислювалися на останніх 120 синхронних точках кожної пари, тобто вже після серії перенавчань і застосування моделей.

Окремо проведено верифікацію на архівній SQLite-базі, що входить до складу проєкту. Вона містить 5 сенсорів, 4280 вимірювань, 142 збережені моделі і 61 позначених аномалій. Цей набір даних є цінним тим, що він показує, як система поводиться не в ідеалізованій симуляції на чотирьох кімнатах, а в наявному архівному сценарії з living-room сенсорами та історією багатьох ітерацій навчання. Для нього проаналізовано останні моделі температурного і вологісного каналів та порівняно їх показники до і після корекції.

Результати дослідження

Початковий запуск batch-калібрування після накопичення 180 історичних пакетів обробив усі 8 парних нееталонних сенсорів без випадків *insufficient_data*. Для температурних каналів середня MAE зменшилася з 1,56 до 0,31 °C, тобто приблизно на 79,8 %. Для вологісних каналів зменшення було скромнішим: з 1,44 до 1,34 %RH (6,9 %). У середньому по всіх 8 парах первинна процедура знизила MAE з 1,50 до 0,83, що еквівалентно 44,9 % покращення. Отже, уже перший запуск моделі дає відчутний ефект для каналів з вираженим афінним дрейфом.

Як показано в табл. 3 і на рис. 5, у завершальному online-вікні середнє зменшення MAE по всіх парних каналах становило 30,4 %.

Таблиця 3

Якість корекції на останньому online-вікні після серії перенавчань

Пара сенсорів	Пари	Статус	MAE до	MAE після	Покращення, %
Вітальня / температура	120	calibrated	1,859	1,060	43,0
Вітальня / вологість	120	calibrated	1,394	1,275	8,5
Спальня / температура	120	calibrated	1,851	1,078	41,8
Спальня / вологість	120	calibrated	1,343	1,126	16,1
Кухня / температура	120	calibrated	1,865	1,065	42,9
Кухня / вологість	120	calibrated	1,179	1,049	11,0
Кабінет / температура	120	calibrated	1,914	1,119	41,6
Кабінет / вологість	120	calibrated	1,640	1,302	20,6

Температурні сенсори демонстрували стабільно високий ефект (1,87 → 1,08 °C, або 42,3 %), а вологісні — помірний (1,39 → 1,19 %RH, або 14,5 %). Така різниця є закономірною: температурний дрейф у симуляторі ближчий до лінійно-афінного перетворення, яке добре описується Huber-Regressor, тоді як вологість має більший стохастичний шум і сильніше насичення, через що виграш від одномірної лінійної корекції менший.

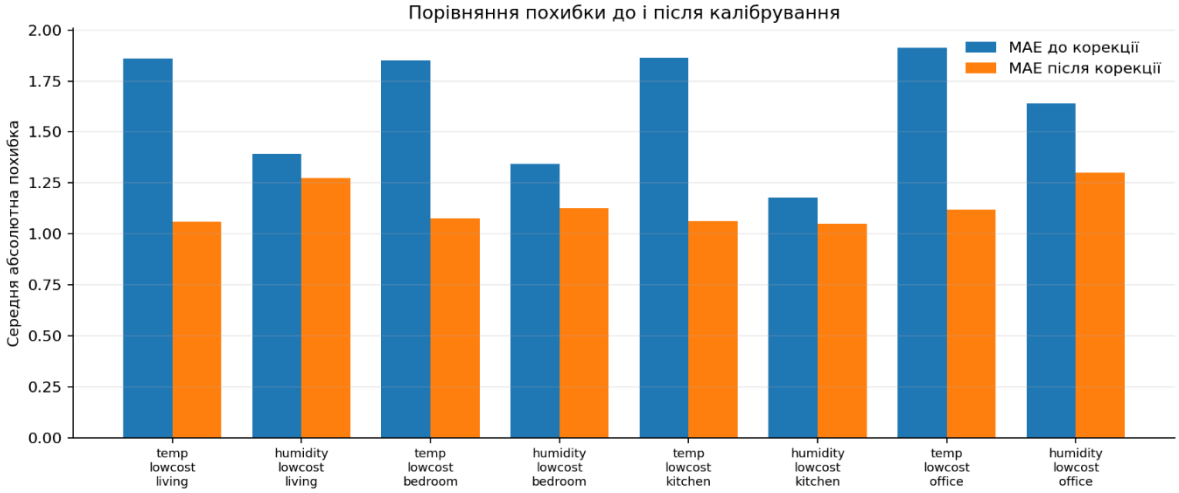

Рис. 5. Порівняння середньої абсолютної похибки до і після корекції для всіх парних сенсорів

На рис. 6 видно, що calibrated-ряд істотно наближається до еталонного, хоча повністю не збігається з ним через безперервне накопичення дрейфу між моментами перенавчання та наявність шуму. Упродовж усього сценарію збережено 48 версій моделей, а в summary наприкінці експерименту зафіксовано 20 сенсорів, 4800 вимірювань, 8 каналів з істотним drift і 4 аномалії. Аномальні події в основному припали на CO_2-сенсори, що узгоджується з логікою симулятора, де саме для CO_2 вводяться короткочасні сплески.

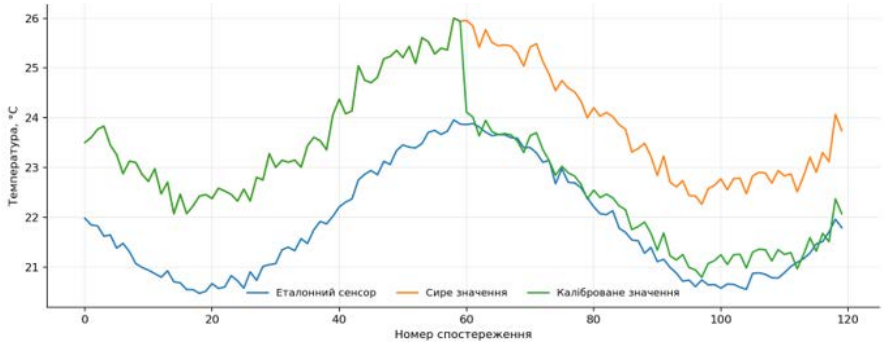

Рис. 6. Приклад онлайн-калібрування для температурного датчика у вітальні

Архівна БД підтверджує ту саму тенденцію на реальному наборі з проєкту (табл. 4). Для *temp_lowcost_living* остання модель знижує MAE з 1,789 до 0,162, тобто майже на порядок, що добре узгоджується з гіпотезою про переважно афінний характер температурного дрейфу. Для *humidity_lowcost_living* поліпшення менше: 0,611 → 0,588, а статус залишається *stable*. З практичної точки зору це корисно: система не нав'язує однаково агресивну корекцію всім сенсорам, а адаптується до реального масштабу похибки.

Таблиця 4

Останні збережені моделі в архівній SQLite-базі проєкту

Архівна пара	MAE до	MAE після	Статус	Інтерпретація
Вітальня / температура	1,789	0,162	calibrated	виражений температурний дрейф, сильна корекція
Вітальня / вологість	0,611	0,588	stable	помірний ефект, канал близький до stable

Обговорення результатів

Розглянута реалізація має кілька сильних сторін. По-перше, вона є інтерпретованою: для кожного сенсора зберігаються коефіцієнти *k* і *b*, зрозумілий drift score, а також MAE до і після корек-

ції. По-друге, сервісна архітектура дозволяє підключати нові джерела даних без зміни ядра алгоритму. По-третє, використання робастної регресії замість звичайної МНК зменшує чутливість до поодиноких помилкових пар, що особливо доречно у побутових IoT-середовищах з нестабільним зв'язком і неоднорідною історією [11].

Водночас варто чітко зафіксувати обмеження. Поточна модель є одномірною і не враховує контекстних факторів, які в сучасних роботах часто підвищують якість калібрування: температуру навколишнього середовища, вологість, швидкість зміни сигналу або міжсенсорні кореляції [1]—[3]. Для вологісних каналів це вже видно з результатів: поліпшення є, але воно набагато нижче ніж для температури. Також відсутні механізми оцінки невизначеності прогнозу та явного керування concept drift, окрім періодичного перенавчання і простого drift score. У висококритичних застосуваннях цього може бути недостатньо.

Перспективними напрямами розвитку є багатовимірна робастна регресія, рекурсивне або online-навчання, а також використання контекстних ознак з інших сенсорів кімнати. Для anomaly detection доцільно порівняти поточний MAD-підхід з легкими моделями класу isolation-based або tree-based detectors, описаними в оглядах IoT anomaly detection [4]. З інженерного погляду варто додати окремий контур повернення скоригованих значень у Home Assistant, щоб corrected stream міг відразу використовуватися в автоматизаціях, а не лише в аналітиці та моніторингу.

Зазначимо, що для зручності користувача розроблено візуальний інтерфейс з використанням платформи Grafana, показаний на рис. 7.

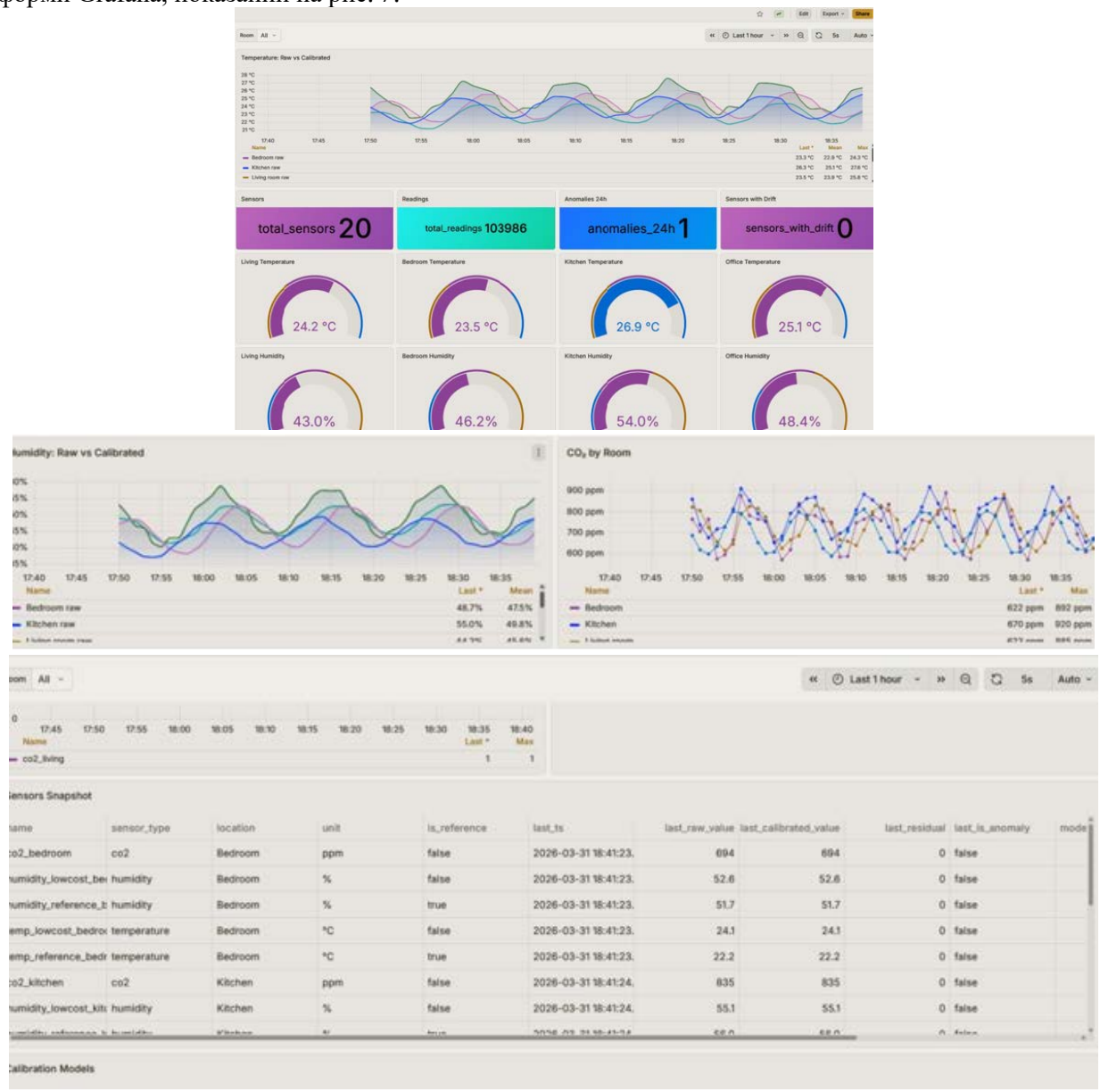

Рис. 7. Інтерфейс користувача

Практичні рекомендації щодо впровадження

З погляду практичного впровадження проєкт уже зараз придатний як зовнішній сервіс корекції для локального smart-home хаба. У типовому сценарії достатньо залишити по одному еталонному температурному та вологісному каналу на кімнату, а дешевші сенсори прив'язати до них через *reference_sensor_id*. Після цього потік показів може надходити у застосунок через MQTT або REST, а скориговані значення використовуватися у дашбордах, правилах автоматики чи модулях аналітики. Кодова база не вимагає втручання у firmware кінцевих пристроїв, тому розгортання можливе поверх уже наявної домашньої інфраструктури.

Для стабільної експлуатації доцільно дотримуватися кількох інженерних правил. По-перше, період перенавчання варто узгоджувати з динамікою дрейфу: для повільних температурних сенсорів інтервал у десятки секунд або хвилини є достатнім, але для каналів зі швидкими змінами може знадобитися адаптивний schedule. По-друге, статус *attention* має використовуватися як сервісний сигнал для оператора або автоматичного правила: якщо drift суттєвий, але модель не проходить критерій поліпшення, значить потрібна перевірка монтажу, стану сенсора або еталонного каналу. По-третє, історія *calibration_models* повинна зберігатися як журнал аудиту, що дозволяє відслідковувати деградацію пристрою в довгому горизонті.

Окремої уваги потребує інтеграція corrected stream у керуючий контур будинку. Для низькоризикових сценаріїв, таких як довгостроковий моніторинг мікроклімату, застосування *calibrated_value* є цілком виправданим уже в поточній версії системи. Для критичніших рішень, наприклад, аварійного керування вентиляцією чи захисту обладнання, варто поєднувати скоригований сигнал із контролем невизначеності, перевіркою свіжості моделі та обмеженнями на максимально допустиму корекцію. Саме така багаторівнева схема дозволяє перетворити дослідницький прототип на виробничий модуль IoT-аналітики.

Загрози валідності та подальша верифікація

Попри отримані позитивні результати, достовірність висновків має кілька потенційних загроз. Перша пов'язана з тим, що основний експеримент виконувався на вбудованому симуляторі, де закон дрейфу заздалегідь наближений до афінного. У реальних будинках залежність між низьковартісним та еталонним каналами може бути складнішою через інерційність сенсора, розташування, локальні теплові потоки чи вплив вентиляції. Друга загроза полягає у використанні лише одного архівного набору з living-room сенсорами; цього достатньо для якісної перевірки працездатності, але недостатньо для статистично сильної генералізації на всі типи приміщень та всі сезони експлуатації.

Для підсилення зовнішньої валідності доцільно організувати довший польовий експеримент з кількома будинками або кімнатами, де еталонні сенсори зберігатимуться впродовж тижнів чи місяців. У такому дослідженні варто окремо аналізувати стабільність коефіцієнтів k і b, частоту спрацьовування status attention, похибку після тривалих перерв у даних та поведінку алгоритму за сезонних зсувів. Додатково корисно провести ablation-study: порівняти HuberRegressor зі звичайною лінійною регресією, RANSAC або деревами рішень, а MAD-детектор — з IQR, EWMA чи lightweight isolation-based методами. Саме така розширена верифікація стане логічним подальшим кроком після описаного прототипу.

Висновки

У статті показано, що розроблений програмний додаток реалізує повноцінну платформу самокалібрування сенсорів розумного будинку, а не лише набір утиліт для демонстрації. Її ядро складається з часового вирівнювання парних каналів, робастної афінної корекції на HuberRegressor, явної логіки прийняття/відхилення моделі та робастної статистичної детекції аномалій. Експериментально підтверджено, що така архітектура особливо ефективна для температурних каналів із систематичним дрейфом, а також забезпечує помірний, але стабільний виграш для вологості. Завдяки сервісній організації, використанню MQTT, FastAPI, PostgreSQL і Grafana рішення може бути розгорнуте як зовнішній локальний модуль без втручання у прошивку сенсорів чи хабу. Отже, запропонований підхід є практично придатною основою для побудови самокаліброваної домашньої IoT-системи з подальшим розширенням до багатовимірних і контекстно-адаптивних моделей.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] K. Yamamoto, T. Togami, N. Yamaguchi, and S. Ninomiya, “Machine Learning-Based Calibration of Low-Cost Air Temperature Sensors Using Environmental Data,” *Sensors*, vol. 17, no. 6, article 1290, 2017. https://doi.org/10.3390/s17061290 .

[2] R. Dubey, A. Telles, J. Nikkel, et al., “Low-Cost CO_2 NDIR Sensors: Performance Evaluation and Calibration Using Machine Learning Techniques,” *Sensors*, vol. 24, no. 17, article 5675, 2024. https://doi.org/10.3390/s24175675 .

[3] M. Taştan, “Machine Learning-Based Calibration and Performance Evaluation of Low-Cost Internet of Things Air Quality Sensors,” *Sensors*, vol. 25, no. 10, article 3183, 2025. https://doi.org/10.3390/s25103183 .

[4] K. DeMedeiros, A. Hendawi, and M. Alvarez, “A Survey of AI-Based Anomaly Detection in IoT and Sensor Networks,” *Sensors*, vol. 23, no. 3, article 1352, 2023. https://doi.org/10.3390/s23031352 .

[5] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata, “Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median,” *Journal of Experimental Social Psychology*, vol. 49, no. 4. pp. 764-766, 2013. https://doi.org/10.1016/j.jesp.2013.03.013 .

[6] OASIS. MQTT Version 5.0. OASIS Standard, 07 March 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html . Accessed: March 17, 2026.

[7] Home Assistant. MQTT integration documentation. [Online]. Available: https://www.home-assistant.io/integrations/mqtt/ . Accessed: February 14, 2026.

[8] *Scikit-learn developers*. HuberRegressor – official API reference. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.HuberRegressor.html. Accessed: February 14, 2026.

[9] FastAPI. Official documentation. [Online]. Available: https://fastapi.tiangolo.com/. Accessed: February 14, 2026.

[10] Grafana Labs. PostgreSQL data source documentation. [Online]. Available: https://grafana.com/docs/grafana/latest/datasources/postgres/ . Accessed: March 17, 2026.

[11] P. J. Huber, “Robust Estimation of a Location Parameter,” *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73-101, 1964. https://doi.org/10.1214/aoms/1177703732 .

Рекомендована кафедрою автоматизації та інтелектуальних інформаційних технологій ВНТУ

Дата надходження 9.04.2026
Дата прийняття до друку після рецензування 18.04.2026
Дата публікації 7.07.2026

Бойко Ольга Володимирівна — студентка факультету робототехніки та приладобудування, e-mail: olia.bojko.98@gmail.com . https://orcid.org/0009-0007-3991-6410.

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ;

Добролюбова Марина Валеріївна — канд. техн. наук, доцент кафедри інформаційно-вимірювальних технологій Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського», Київ; доцент кафедри інформаційних систем в економіці, Київського національного економічного університету імені Вадима Гетьмана, Київ, e-mail: m.dobroliubova@ukr.net . https://orcid.org/0000-0003-3647-3320

O. V. Boiko[1]
M. V. Dobroliubova[1, 2]

Smart-Home Sensor Self-Calibration System Based on Robust Regression and Online Anomaly Detection

[1]National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute»;
[2]Kyiv National Economic University named after Vadym Hetman

The paper examines a smart-home sensor self-calibration platform implemented as an external Python service for local Internet of Things (IoT) infrastructures. The relevance of the study is determined by the fact that drift and systematic bias of low-cost temperature and humidity sensors directly affect indoor climate monitoring, the correctness of automated heating and ventilation scenarios, and the overall energy efficiency of residential environments. At the software level, the study reconstructs a complete processing pipeline that covers data

ingestion through a REST API, MQTT or a built-in simulator, persistence in PostgreSQL, periodic retraining of the calibration model, correction of incoming measurements, online anomaly detection, and subsequent visualization through a web dashboard and Grafana. The methodological novelty of the solution lies in combining time alignment of "raw sensor – reference sensor" pairs, robust affine correction with HuberRegressor, a model acceptance rule based on reducing mean absolute error (MAE) by at least 2%, and anomaly scoring based on median absolute deviation (MAD) with a fallback z-score procedure. The study shows that the algorithm does not merely train a correction model; it can also safely refuse to apply calibration when the model quality is insufficient or when the amount of training data is inadequate. An experiment performed on the built-in simulator, which models four rooms, 20 sensors and 4,800 measurements, demonstrated a strong error reduction for temperature channels: after the initial calibration stage, the average MAE decreased from 1.56 to 0.31 °C. For humidity channels, the effect was more moderate, with the average error decreasing from 1.44 to 1.34 % RH. In the final online window after a series of retraining cycles, the mean MAE reduction across all paired channels reached 30.4 %; specifically, temperature sensors achieved a 42.3 % improvement, whereas humidity sensors achieved a 14.5 % improvement. Additional verification on the archived SQLite database bundled with the project confirmed a strong effect for the living-room temperature sensor (1.789 → 0.162) and a moderate effect for the humidity channel (0.611 → 0.588). The practical significance of the proposed approach lies in the fact that it can be integrated into the existing local smart-home infrastructure without modifying the firmware of end devices, thereby improving telemetry reliability, reducing false automation triggers, and creating prerequisites for energy savings in home automation systems. Beyond its technical effect, the proposed solution also has clear applied socio-economic significance, since it makes it possible to improve the performance of low-cost sensors by software means without switching to more expensive hardware. This can substantially simplify everyday use for people relying on affordable smart-home systems, make such systems more accessible to a wider range of consumers, and, in the long term, contribute to reducing their overall cost both at the deployment stage and during further operation.

Keywords: smart home, sensor self-calibration, robust regression, HuberRegressor, mean absolute error, median absolute deviation, anomaly detection, MQTT, FastAPI, PostgreSQL, Grafana, programming, database.

Boiko Olha V. — Student of the Department of Robotics and Instrumentation, e-mail: olia.bojko.98@gmail.com . https://orcid.org/0009-0007-3991-6410;

Dobroliubova Maryna V. — Cand. Sc. (Eng.), Associate Professor of the Chair of Information and Measurement Technologies of Igor Sikorsky Kyiv Polytechnic Institute, Kyiv; Associate Professor of the Chair of Information Systems in Economics of Vadym Hetman Kyiv National Economic University, Kyiv, e-mail: m.dobroliubova@ukr.net . https://orcid.org/0000-0003-3647-3320