Одеський національний університет імені І. І. Мечникова

Факультет математики, фізики та інформаційних технологій

Кафедра оптимального керування та економічної кібернетики

Кваліфікаційна робота

на здобуття ступеня вищої освіти «магістр»

«Аналіз часових рядів для порівняння метрик інформаційної системи»
«Time series analysis for information system metrics comparing»

Виконав: здобувач денної форми навчання
спеціальності 113 Прикладна математика
Освітня програма «Прикладна математика»

Воротов Дмитро Вадимович

Керівник

ст. викл. Платонов В.В.

(науковий ступінь, вчене звання, прізвище та ініціали, підпис)

Рецензент

доктор фіз.-мат. наук, доцент, Кічмаренко О.Д.

(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:

Протокол засідання кафедри

№ ___ від _____2024 р.

Завідувач кафедри

_____ _____

(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК №_____

протокол № ____ від _____ 2024 р.

Оцінка_____/___ ___/_____

(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ _____

(підпис) (прізвище, ініціали)

Одеса – 2024

ЗМІСТ

ВСТУП

У сучасному світі інформаційні системи стали невід'ємною складовою бізнесу, науки та повсякденного життя. Розуміння стану інфраструктури та систем є критично важливим для забезпечення надійності та стабільності роботи сервісів. Інформація про працездатність та продуктивність розгортань не лише допомагає команді реагувати на проблеми, але й дає впевненість для внесення змін. Одним із найкращих способів отримати таке розуміння є надійна система моніторингу, яка збирає метрики, візуалізує дані та сповіщає операторів, коли щось виходить з ладу [1].

Проте у великих інформаційних системах кількість таких метрик може досягати тисяч чи навіть сотен тисяч, що робить їхнє візуальне відстеження практично неможливим. Ручний аналіз та візуалізація великої кількості метрик стають неефективними та неоперативними методами виявлення проблем. Оскільки кожна інформаційна система має унікальний набір метрик, які суттєво варіюються між системами, точне та своєчасне відстеження цих метрик є критично важливим для стабільної роботи системи. Це створює необхідність у розробці автоматизованих методів аналізу, які можуть швидко та ефективно обробляти великі обсяги даних. Загалом, таку задачу можна вирішувати 3 напрямами - reactive (постфактум), predictive (прогнозування) або perspective (рекомендації оптимального курс дій), це дослідження ближче до reactive та на межі з predictive.

Аналіз часових рядів, тобто аналіз змін метрик у часі, дозволяє виявляти тренди, сезонні коливання та аномалії, що допомагає ідентифікувати потенційні проблеми в системах. В умовах обмежених обчислювальних ресурсів, коли одночасно потрібно обробляти в реальному часі великі обсяги часових рядів, стає необхідним використовувати максимально ефективні, прості та швидкі методи аналізу. Прості статистичні підходи дозволяють автоматизувати процес моніторингу та швидко отримати огляд основних трендів у даних з

мінімальними витратами ресурсів. Вони забезпечують достатню точність для багатьох прикладних задач, що робить їх придатними для широкого використання в умовах різноманіття систем та метрик.

Сучасні дослідження в галузі аналізу часових рядів активно розвиваються, приділяючи особливу увагу створенню нових методів виявлення аномалій та покращення точності прогнозів. Хоча застосування машинного навчання відкриває перспективи для більш складного аналізу, універсальні та менш ресурсозатратні методи є більш практичними для використання в різних інформаційних системах, особливо коли йдеться про обробку великої кількості метрик. У дипломній роботі акцент зроблено саме на розробці універсальних підходів до аналізу часових рядів, які можуть бути застосовані в умовах обмежених ресурсів для різноманітних метрик, автоматизуючи процес аналізу даних моніторінга та виявлення аномалій без необхідності візуалізації кожної метрики окремо.

Таким чином, аналіз часових рядів є одним з ключових інструментом для забезпечення стабільної та ефективної роботи інформаційних систем. Він дозволяє не лише виявляти і усувати поточні проблеми, але й прогнозувати майбутні зміни, що сприяє підвищенню надійності та продуктивності системи в цілому. Автоматизація процесу аналізу метрик і використання ефективних методів дозволяють обробляти тисячі показників без значних витрат ресурсів, що є особливо важливим у масштабних системах.

РОЗДІЛ 1

ПОСТАНОВКА ЗАДАЧІ, МОНІТОРИНГ І ЗБІР ДАНИХ

1.1 Постановка задачі

Інформаційні системи стають все більш складними, збільшуючи обсяги метрик для моніторингу, що робить ручний аналіз цих даних неефективним. Сучасні методи аналізу часових рядів дозволяють виявляти тренди, сезонні коливання та аномалії, забезпечуючи більш глибоке розуміння продуктивності та стабільності систем. Однак, із зростанням кількості даних виникає необхідність у розробці автоматизованих, універсальних і малозатратних підходів для аналізу великих масивів часових рядів.

Мета цього дослідження – це розробити огляд ефективних та універсальних методів аналізу часових рядів для автоматизованого порівняння та оцінки метрик інформаційних систем, що дозволить виявляти аномалії, покращувати продуктивність та стабільність системи за рахунок оптимального використання обчислювальних ресурсів.

Об'єкт дослідження – метрики інформаційних систем, які змінюються з часом та впливають на їхню продуктивність і надійність.

Предмет дослідження – методи та алгоритми аналізу часових рядів, які дозволяють автоматизувати процес моніторингу, порівняння та виявлення аномалій у великій кількості метрик різних інформаційних систем.

Методи дослідження – статистичний аналіз та інші підходи для аналізу часових рядів, з фокусом на універсальні й малозатратні методи для широкого застосування.

Конкретні завдання для досягнення цієї мети включають:

1. Провести аналіз існуючих методів збору, зберігання та аналізу метрик інформаційних систем та систем моніторингу.
2. Розробити алгоритми автоматичної класифікації часових рядів за їх характеристиками (стабільність, сезонність, трендовість тощо).
3. Розробити підходи до аналізу спираючись на класифікацію
4. Дослідити сучасні методи виявлення аномалій, включаючи Z-score, Local Outlier Factor (LOF) та матричний профіль.
5. Оцінити ефективність запропонованих підходів на реальних даних, зібраних із систем моніторингу.
6. Підсумувати рекомендації для впровадження запропонованих методів у інформаційних системах.

1.2 Моніторинг

Розуміння стану інфраструктури та працездатності систем має важливе значення для забезпечення надійності та стабільності будь-яких сервісів. Для цього створюються системи моніторингу.

Моніторинг базується на зборі, зберіганні, аналізі та візуалізації метрик, логів, трейсів та іншої інформації, яка представляє кількісні та якісні характеристики стану системи. Наприклад, час відгуку сервера, рівень використання оперативної пам’яті, кількість одночасних підключень тощо. В цій роботі розглядаються метрики, які були зібрані системою моніторингу Prometheus. [1]

Prometheus — це одна з найпоширеніших платформ для збору, обробки та аналізу метрик. Вона є відкритим програмним забезпеченням і працює з багатовимірною моделлю даних, що дозволяє ефективно організувати зберігання часових рядів. Особливостями Prometheus є:

- Мультидимензійна модель даних: метрики визначаються за допомогою імені та набору ключів-значень.
- PromQL: Потужна мова запитів для роботи з часовими рядами, що дозволяє легко виконувати вибірки, агрегацію та інші операції над даними.
- Автономність: кожен вузол сервера Prometheus є самостійним і не залежить від мережевого сховища.
- Метод збору метрик: використовується pull-модель через HTTP.
- Підтримка push**:** Дані можуть надходити через посередницький шлюз для короткострокових завдань.
- Метрики збираються з цілей, які можуть бути налаштовані статично або виявлятися автоматично через сервіс діскавері.

Також моніторинг може включати зберігання історичних даних, оповіщення та алертингові системи. [2]

1.3 Збір даних

В рамках цієї роботи, так і загалом, дані інформаційних систем можна поділити на три основні категорії:

1. *Метрики інфраструктури:* наприклад, завантаженість процесора (CPU), обсяг вільної оперативної пам’яті, використання дискового простору, тривалість запитів, метрики API тощо.
2. *Бізнесові метрики:* пов’язані з роботою самого бізнесу, наприклад, кількість активних користувачів, та будь-які інші метрики, обрані бізнесом.
3. *Логи помилок та відповідні метрики:* наприклад, кількість помилок або подій, які призводять до краху системи.

Дані для метрик можуть походити з різних джерел, включаючи:

- *Серверні метрики:* відображають стан серверів, інфраструктури (системи віртуалізація та контейнеризації) тощо, таких як CPU, пам'ять, дискове використання.
- *Метрики додатків:* містять інформацію про роботу програмного забезпечення, включаючи успішність виконання запитів, затримки, кількість помилок.
- *Мережеві метрики:* характеризують стан мережевої взаємодії, включаючи доступність, пропускну здатність і затримки.
- *Дані зовнішніх залежностей:* охоплюють стан зовнішніх сервісів, які можуть впливати на роботу системи.

Дані для цієї роботи були отримані з реальної системи у форматі JSON, де кожна метрика супроводжується мета інформацією, яка описує її параметри та контекст. В результаті кожна метрика – це її контекст (метаінформація) там на набір пар час - значення метрик. Наприклад:

```
{
 "parameters": {
```

```
    "run": {...},
    "metadata": {"datasource": {...}},
    "metrics": [{*metric metadata*}, ...]
  },
  "data": [{ "cpu usage percent": {
        "status": "success",
        "data": {
          "resultType": "matrix",
          "result": [
            {
              "metric": {
                "workload": "iat-worker",
                "workload_type": "deployment"
              },
              "values": [[1730192895.561, "0.010742828786425783"]]
}]}}}]}
```


Усі зібрані метрики агрегуються в загальний датасет (табл. 1.3.1), який має часову шкалу з рівномірним інтервалом. Кожен запис у цьому датасеті описує значення метрики в конкретний момент часу разом із додатковими полями, такими як ім'я метрики, унікальний ідентифікатор та інші.

Таблиця 1.3.1

Датасет з часовими рядами

timestamp	**value**	**metric_name**	**batch**	**metric_id**
2024-09-18 08:20:00	0.00	tpi response size	0	0a8108389bb
2024-09-18 08:21:00	0.01	tpi response size	0	0a8108389bb
2024-09-18 08:20:00	0.0621	tpi response size	0	247a2bd357

Метаінформація про кожну метрику зберігається в окремому датасеті (табл. 1.3.2). У ньому зберігається інформація про джерело метрики, та вся її додаткова інформація. З цієї інформації і будується унікальний ідентифікатор. Наприклад, дві метрики використання CPU можуть мати однакову назву, але

різні *metric_id*, оскільки вони належать до різних нод серверу і можуть мати абсолютно різний характер поведінки тощо (наприклад, рис. 2.2.2, рис. 3.1.1, і рис. 3.2.3 описують різні метрики CPU). Надалі під ***метрикою*** будемо розуміти унікальний часовий ряд відповідний до його *metric_id,* а під ***групою метрик*** - всі метрики із спільною назвою *metric_name.*

Таблиця 1.3.2

Датасет з метаінформацією метрик

metric_id	**metric name**	**…**			**instance**	**namespace**
0a8108389bb	tpi response size	…	…	…	localhost:8080	aap-awspgsite
247a2bd357	tpi response size	…	…	…	localhost:8080	aap-prod

В результаті у виборці для цієї роботи є 52 групи метрик із 1980 унікальними метриками - це дані реального сервісу, якій складається лише з одного вузла. В реальних системах це десятки сервісів, систем баз даних, систем віртуалізації та контейнеризації і кількість метрик може сягати сотен тисяч.

Також у процесі збору даних виявилися можливі нюанси в даних:

1. *Різні порядки величин* - метрики використання CPU, можуть варіюватися від часток відсотка до сотень відсотків, тоді як метрики, такі як "jvm heap utilization", можуть досягати мільйонів чи навіть мільярдів. Це ускладнює порівняння даних без відповідної нормалізації.
2. *Різні кроки часу* - не всі метрики прив'язані до загальної часової шкали з рівномірним кроком шкали.
3. *Аналіз подій:* Метрики, пов’язані з помилками або крахами, здебільшого мають характер івентів, тобто можуть бути представлені як набори подій, а не як безперервні часові ряди. [3]

4. *Різні розміри часових рядів* - розміри часових рядів у виборці змінюються від 2-6 точок івентів, до 5000 регулярних значень.

Наразі єдина спільна характеристика по всім метрикам - це визначення даних, як часових рядів, та наявність певних метаданих із системи. При цьому всі метадані будуть знову таки унікальними для кожної інформаційної системи.

РОЗДІЛ 2

КЛАСИФІКАЦІЯ ЧАСОВИХ РЯДІВ

2.1 Модель класифікації

Наступний кроком роздивимося навіщо та як буде здійснюватися класифікація метрик. Як було зазначено, система може містити сотні тисяч метрик (часових рядів) і детальний аналіз кожної є майже неможливим. Однак їх можна автоматичко класифікувати за типами часових рядів. Тоді їх класифікація дозволить виявляти аномалії всередині груп метрик, що суттєво зменшить обсяг майбутньої роботи та заощадить ресурси. Крім того, це спростить пошук аномалій у майбутньому. Роздивимося властивості, за якими будем класифікувати:

1. Наявність даних (*is_data_lack*) – перевірка розміру часового ряду, бо при замалих об'ємах даних будет неможливо щось стверджувати.
2. Константність (*is_constant*) – визначення, чи є ряд постійним (або значення майже не змінюється). Постійні ряди можуть бути малокорисними для подальшого аналізу, але їх зміни можуть бути важливими тригерами.
3. Стаціонарність (*is_stationary*), наявність тренду (*is_trend*) та сезонність/циклічність (*is_seasonal*) – аналіз статистичних властивостей ряду. Це допомагає зрозуміти, чи є дані передбачуваними, чи містять вони повторювані патерни або довгострокові зміни.

На жаль, якість класифікації залежатиме від обраних нами гіперпараметрів. Ці параметри, ймовірно, залишаться такими, що налаштовуються вручну. Однак в результаті, цей підхід дозволить зменшити кількість рядків для роботи з мільйонів до близько 2000. Це значно знизить обсяг обчислень, спростить роботу із системою і зробить аналіз аномалій більш ефективним.

2.2 Константність та стаціонарність

Під константністю ряду будемо розуміти процент значень, який займає мода ряду (значення випадкової величини, що трапляється найчастіше в сукупності спостережень). Для суцільно константного ряду - це, зрозуміло, 100%, але досить часто можна зустріти ряди, які константні більшу частину часу і іноді мають викиди (рис. 2.2.1), що може бути як аномалією, так і нормою. Далі будемо відносити такі ряди теж до константних, а от межа константності тоді утворює новий гіперпараметр системи аналізу.

Рис 2.2.1 Не абсолютно константний ряд

Також з цим же кроком будет проста перевірка розміру часового ряду і якщо він менше обраного порогу (N точок), то подальший аналіз і класифікація не відбуватиметься, з нестачі даних. Цей крок необхідний для роботи більшості розглянутих алгоритмів, які потребують для роботи певного обсягу даних.

Стаціонарний часовий ряд — це такий, статистичні характеристики якого не залежать від часу спостереження. Більш точно, якщо y_t є стаціонарним часовим рядом, то для всіх s розподіл $(y_t, \dots, y_{t+s})$ не залежить від t. [4]

Для перевірки стаціонарності часового ряду використовується тест Дікі-Фуллера (Augmented Dickey-Fuller Test, ADF). Цей тест перевіряє наявність одиничного кореня у процесі, що є свідченням нестабільності ряду.

Тест оцінює наступне рівняння:

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \delta_1 \Delta y_{t-1} + \cdots + \delta_p \Delta y_{t-p} + \epsilon_t \ ,$$

де:

- Δy_t — перша різниця ряду $(y_t - y_{t-1})$
- α — константа,
- βt — тренд у ряді,
- $\delta_1 \Delta y_{t-1}$ — коефіцієнт перед лаговим значенням, який визначає наявність одиничного кореня,
- ϵ_t — випадкова похибка,
- p — кількість лагів, що враховуються для зняття автокореляції.

Якщо значення статистики γ є статистично значущим (тобто p-значення менше обраного рівня значущості, зазвичай 0.05), ми відхиляємо нульову гіпотезу і вважаємо, що ряд стаціонарний.[5]

Рівень значущості - це наступний гіперпараметр системи класифікації.

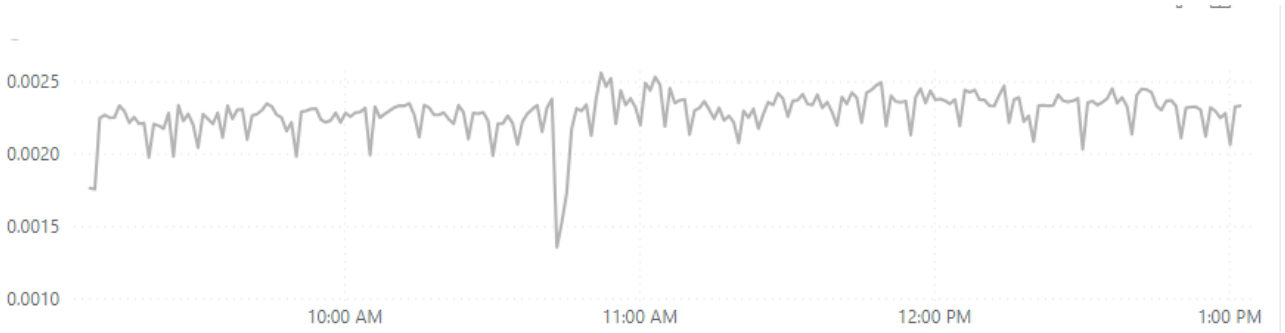

Рис 2.2.2 Приклад стаціонарного ряду з виборки

Згідно з теорією, стаціонарний часовий ряд не може мати тренду або сезонності. Проте, реальності виявляється складнішою через випадкові похибок, якість даних, обмеження самого тесту тощо. Через це навіть ряди, які визналися тестом стаціонарними, будуть перевірятися на наявність трендів чи сезонності. При цьому, протиріччя, наприклад, коли ряд класифікується як стаціонарний разом з трендом, можуть слугувати індикаторами для покращення гіперпараметрів моделі або ж бути індикаторами можливих аномалій чи структурних змін у даних, які могли вплинути на результати тесту.

2.3 Естімейт сезонності

Після перевірки стаціонарності часового ряду наступним кроком є визначення тренду та циклічності. Це включає виділення компонентів ряду за допомогою STL-декомпозиції або інших методів, попередньо необхідно оцінити довжину періоду. Цей процес побудуємо на частотному (спектральному) та кореляційному аналізі.

Будь-який часовий ряд можна подати як комбінацію синусів і косинусів із різними періодами (тривалістю одного циклу) та амплітудами (максимальними і мінімальними значеннями в циклі). Ця властивість використовується для дослідження періодичної поведінки ряду. [6]

Періодограма дозволяє перетворити часовий ряд на частотне розподілення. Це допомагає визначити домінантні частоти, які відповідають найсильнішим повторюваним компонентам (рис. 2.3.2, правий). Виділяючи найбільші піки частот за допомогую будь-якого алгоритму отримаємо потенційні періоди циклів в даних (якщо спектр показує пік на частоті f, то відповідний період можна обчислити як $1/f$ та округлити).

Щоб підтвердити результати спектрального аналізу, використовується автокореляційна функція (ACF), яка оцінює зв’язок між значеннями ряду на різних лагах. Повторювані піки автокореляції вказують на періодичність у ряді. Та й сама функція автокореляції повинна зберігати циклічність.

Поєднання періодограми та ACF дозволяє визначити сезонний період більш точно. Якщо період, знайдений у спектрі, співпадає з одним із піків ACF або має мінімальні розбіжності або повністю повторюється, цей період можна вважати достатнім для подальшого використання STL.

Слід зазначити, що спектральний аналіз можу бути більш вигідним на великих об’ємах даних, бо із швидким перетворенням Фур'є (FFT) асимптотика будет $O(n * log(n))$, в таких випадках ACF може бути більш дорожчим. Та в

цьому випадку можна використати й інші методи, бо в наступних кроках буде більш точна перевірка циклічності.

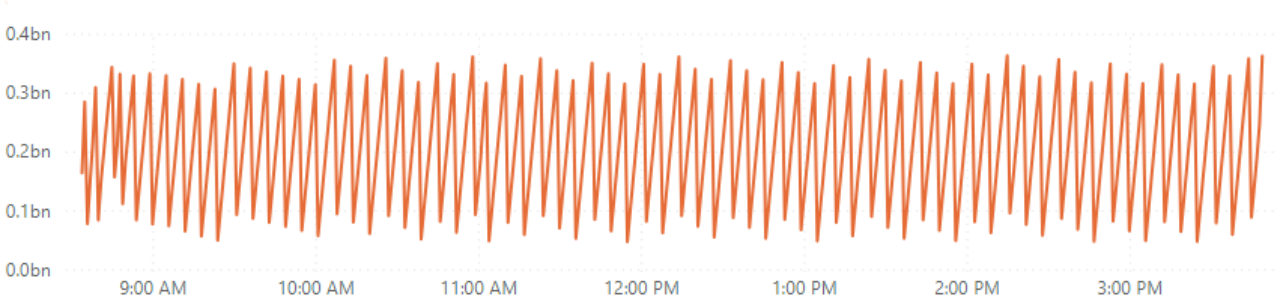

Рис 2.3.1 Приклад циклічного ряду JVM heap utilization

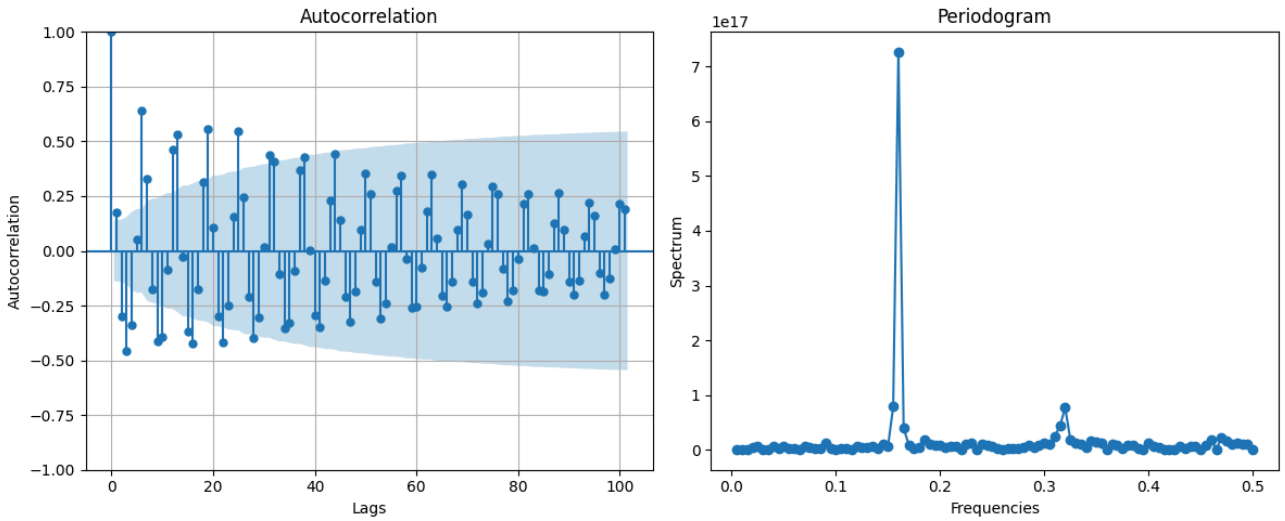

Рис 2.3.2 Автокореляція та періодограма для рис. 2.3.1 з періодом 6 точок

2.4 STL

STL-декомпозиція (Seasonal-Trend Decomposition using LOESS) є потужним методом, що дозволяє розділити часовий ряд на три компоненти: тренд, сезонність і залишкову частину. Цей підхід забезпечує гнучкість і адаптивність, особливо для аналізу рядів із нерегулярною або слабкою сезонністю. Для успішної роботи STL важливо попередньо оцінити довжину сезонного періоду, що визначається за допомогою частотного та кореляційного аналізу.[7]

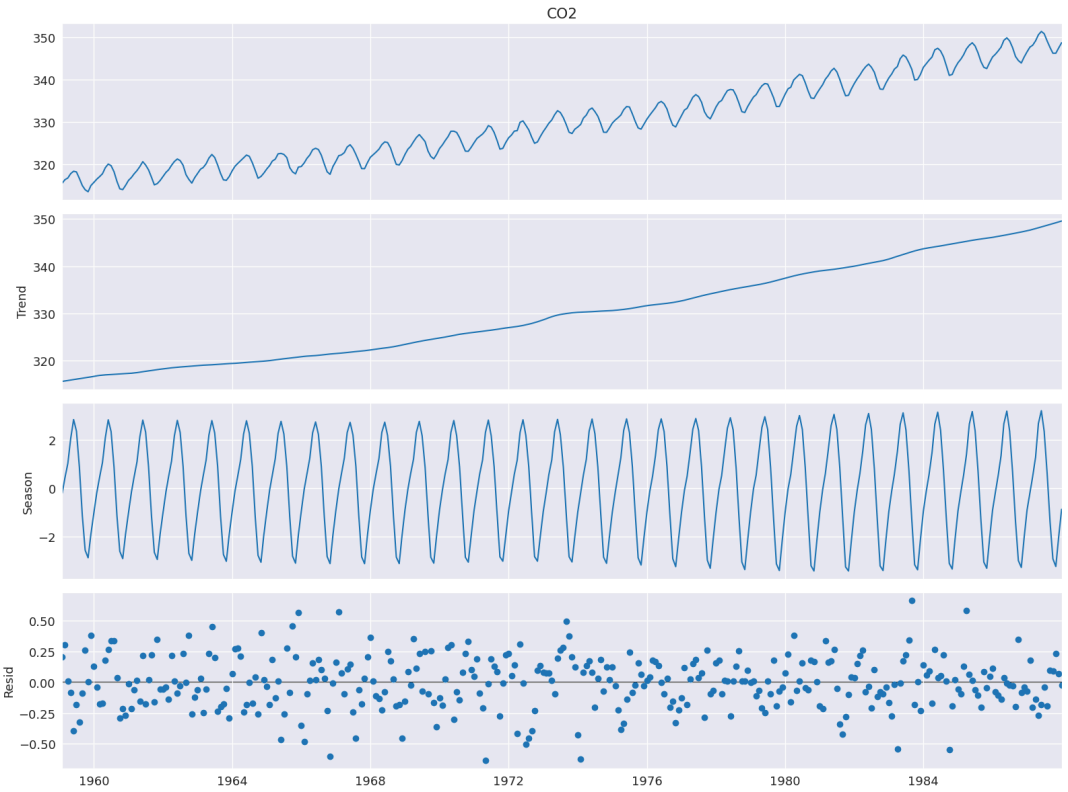

Рис 2.4.1 Приклад STL-декомпозиція [9]

STL розділяє часовий ряд на:

- Трендову компоненту, яка відображає довгострокову зміну даних;
- Сезонну компоненту, що містить регулярні повторювані цикли з обраним періодом;
- Залишкову компоненту, яка включає нерегулярні коливання, шум і аномалії.

Ці компоненти оцінюються за допомогою локальної регресії (LOESS), яка дозволяє гнучко адаптуватися до особливостей даних. [8]

Сезонність вважається суттєвою, якщо варіація сезонної компоненти значно перевищує варіацію залишкової. Для кількісної оцінки використовується сила сезонності. Трендовість оцінюється за допомогою сили тренду, яка розраховується аналогічно сезонності:

$$S = max(0, 1 - \frac{Var(residuals)}{Var(residuals)+Var(seasonality)})$$

$$T = max(0, 1 - \frac{Var(residuals)}{Var(residuals)+Var(trend)})$$

Де:

- $Var(residuals)$ — дисперсія залишкової компоненти,
- $Var(seasonality)$ — дисперсія сезонної (циклічної) компоненти,
- $Var(trend)$ — дисперсія трендової компоненти. [7]

Сезонність вважається суттєвою, якщо $S > \{\text{межа сезонності}\}$, в нашому випадку було обрано 0.5, але це знов так гіперпараметр. Це значення може змінюватися залежно від природи даних. Тренд вважається значущим, якщо $T > \{\text{межа сили тренду}\}$. Знов таки це гіперпараметр. Для часових рядів із високим рівнем шуму цей поріг може бути вищим.

Залишкова компонента відіграє ключову роль у розрахунках сили сезонності та тренду. Високий рівень шуму або нерегулярні аномалії в залишках можуть знижувати оцінку сили сезонності (S) і тренду (T), навіть якщо ці компоненти насправді присутні. Тому результати STL-декомпозиції слід інтерпретувати з урахуванням характеристик залишкової компоненти. Низькі значення сили компонент ($S \approx 0$ і $T \approx 0$) можуть свідчити як про випадковий характер даних, так і про слабко виражені закономірності, тому ці межі потребують тонкого налаштування.

2.5 Аналіз за класифікацією

Як зазначалося раніше, під час побудови алгоритму класифікації ми отримували деякі гіперпараметри:

- *data_lack_qty (=50 - значення, використанне в роботі)* - межа нестачі даних в часовому ряді, для подальшого аналізу і класифікації ряд повинен мати більшу кількість точок ніж ця межа.
- constant_lvl (=0.99) - цей параметр визначає межу для класифікації ряду як константного, константні ряди теж не потрапляють у подальшу класифікацію.
- conf_a (=0.05) - параметр визначає рівень значущості для тесту Дікі-Фуллера (ADF), який перевіряє ряд на стаціонарність.
- trend_lvl (=0.3) - параметр використовується для визначення трендовості ряду. Він задає порогове значення сили тренду.
- seasonal_lvl (=0.5) - визначає поріг для сили сезонності, зауважимо, що за цим порогом естімірований період циклічності може бути відхиленим.

Також треба зазначити “сховані” гіперпараметри, а саме: алгоритм визначення стаціонарності, алгоритм визначення період циклу, алгоритм декомпозиції. Їх зміна також буде впливати на результати класифікації.

Тепер, визначивши типи метрик як часових рядів, нам відкриваються можливості аналізувати аномалії в самих метриках, відхилення метрик від їх класу та інші інсайти з оновленої метаінформації метрик.

Роздивимося загальний розподіл метри по класам (рис 2.5.1):

- ≈26% даних потрапляють в категорію “нестачі даних”,
- ≈20% визначені константами, але тільки 8% перетнули межу нестачі,
- ≈44% є суто стаціонарними, що трохи спрощує пошук аномалій серед них
- трохи більше 1% потрапили до стаціонарних з трендом, що є непоганим результатом, але може бути простір до налаштувань гіперпараметрів
- так само 2% потрапили до стаціонарно-сезонних,

- ≈18% не класифікувалися і потребують більше детального аналізу, наприклад диференціюваня. Також серед них помітна концентрація бізнесових метрик, які можуть сильно залежати від людського фактору як включення чи виключення сервісу тощо.

is_data_lack	is_constant	is_stationary	is_seasonal	is_trend	Metrics QTY	%
					355	17.93%
				✓	1	0.05%
			✓		3	0.15%
			✓	✓	9	0.45%
		✓			867	43.79%
		✓		✓	11	0.56%
		✓	✓		39	1.97%
		✓	✓	✓	14	0.71%
	✓				163	8.23%
✓					293	14.80%
✓	✓				225	11.36%
Total					**1980**	**100.00%**

Рис 2.5.1 Розподіл метрик за класифікацією

Загальний аналіз наводить на думки про потенційні доопрацювання, але якщо роздивитись розподіл всередині груп метрик, то ми зможемо робити висновки більш детально і все зараз виявляти метрики, що помітно виділяються від своїх груп (рис. 2.5.2, потенційні відхилення виділені знаком оклику).

А додавши метаінформацію про певні метрики з урахуванням контексту системи та метрик, людина, що розуміє значення метаінформації, може скоригувати потрібні діменшини і розрахунки. Таким чином побудувавши більш коректну систему моніторингу чи дашборд.

is_data_lack	metric_name	is_constant	is_stationary	is_seasonal	is_trend	Metrics QTY	%_in_group
False	MVC JVM Heap Utilization bytes					3	3%
False	MVC JVM Heap Utilization bytes		✓			31	35%
False	MVC JVM Heap Utilization bytes		✓		✓	5	6%
False	MVC JVM Heap Utilization bytes			✓		3	3%
False	MVC JVM Heap Utilization bytes		✓	✓		39	44%
False	MVC JVM Heap Utilization bytes		✓	✓	✓	8	9%
False	AAP Wallet Latency MoneyTransactionAvgP95 total ms		✓			1	100%
False	AAP Wallet Latency MoneyTransactionAvgP95 ms					101	55%
False	AAP Wallet Latency MoneyTransactionAvgP95 ms		✓			79	43%
False	AAP Wallet Latency MoneyTransactionAvgP95 ms	✓				2	1%
False	AAP service2 tpi transaction processing errors count		✓			3	27%
False	AAP service2 tpi transaction processing errors count	✓				8	73%
False	AAP service2 tpi response size		✓			3	27%
False	AAP service2 tpi response size	✓				8	73%
False	AAP service2 network usage transmit bandwidth in bytes		✓			15	75%
False	AAP service2 network usage transmit bandwidth in bytes		✓		✓	4	20%
False	AAP service2 network usage transmit bandwidth in bytes		✓	✓	✓	1	5%
False	AAP service2 network usage received bandwidth in bytes		✓			20	100%
False	AAP service2 memory usage in bytes					8	40%
False	AAP service2 memory usage in bytes		✓			11	55%
False	AAP service2 memory usage in bytes		✓		✓	1	5%
False	AAP service2 cpu usage percent					1	5%
False	AAP service2 cpu usage percent		✓			19	95%
False	AAP BUSINESS StartGameSession Total Count		✓			1	100%
False	AAP BUSINESS StartGameSession Count		✓			20	100%
False	AAP BUSINESS MoneyTransactions Count		✓			1	100%
False	AAP Business Latency PlayerSessionAvgP95 total ms					1	100%
False	AAP Business Latency PlayerSessionAvgP95 ms					72	40%
False	AAP Business Latency PlayerSessionAvgP95 ms		✓			106	59%
False	AAP Business Latency PlayerSessionAvgP95 ms		✓	✓	✓	1	1%
False	AAP Business Latency PlayerSessionAvgP95 ms	✓				1	1%
False	AAP Business Latency GetPlayerInfoAvgP95 total ms					1	100%
False	AAP Business Latency GetPlayerInfoAvgP95 ms					89	52%
False	AAP Business Latency GetPlayerInfoAvgP95 ms		✓			78	46%
Total						**1462**	**100%**

Рис 2.5.2 Розподіл класів всередині груп метрик

РОЗДІЛ 3

АНОМАЛІЇ

3.1 Стандартизована оцінка

Маючи результати класифікації та STL декомпозиції, можемо визначити швидкий та простий метод виявлення аномалій:

1. Для константих рядів аномалією будемо вважати відхилення від моди ряду (те саме константне значення).
2. Для усіх інших будем використовувати результати STL декомпозиції (аналіз стаціонарних рядів співпадає з аналізом залишків) і метод стандартного відхилення.

Z-score (стандартне відхилення) — це метрика, що вимірює, наскільки далеко точка даних знаходиться від середнього значення ряду у кількості стандартних відхилень. Він розраховується за формулою:

$z = \frac{X - \mu}{\sigma}$, де:

- X — значення окремої точки,
- μ — середнє значення ряду,
- σ — стандартне відхилення ряду.

Зазвичай обирають дані, які знаходяться на відстані більше 2 або 3 стандартних відхилень від середнього залишку, і класифікуються як аномалії. Але для більш гнучкого налаштування, відстань *z_threshold* перейде до гіперпаметрів системи.

Повернемося до STL-декомпозиції. Залишкова компонента STL-декомпозиції представляє ту частину даних, яку неможливо пояснити за допомогою сезонної та трендової компонент. Ідеально, залишки мають бути

випадковим шумом. Проте у практичних задачах аномалії часто проявляються як незвичні стрибки або падіння у залишковій серії, які значно відхиляються від типового рівня шуму. Те саме стосується стаціонарних рядів.

Отже для залишків та стаціонарних рядів Z-score обчислюється без додаткових трансформацій, виявляючи значення, які відрізняються від середнього. Також аномалію можна зустріти у аномаліях трендовій і сезонній компонентах, які потрібно виділити перед аналогічним використанням стандартного відхилення способом.

Окремо виділимо нестаціонарні ряди, які не були класифіковані навіть після STL-декомпозиції. Такі ряди піддаються диференціюванню до досягнення стаціонарності.

Диференціювання полягає у взятті різниці між сусідніми значеннями ряду, що дозволяє усунути довгострокові тенденції чи ефекти накопичення. Після цього до отриманих значень застосовується Z-score для виявлення аномалій. Якщо після кількох рівнів диференціювання ряд все одно не стає стаціонарним, це може бути свідченням більш складної структури чи наявності аномалій у самій моделі. [10]

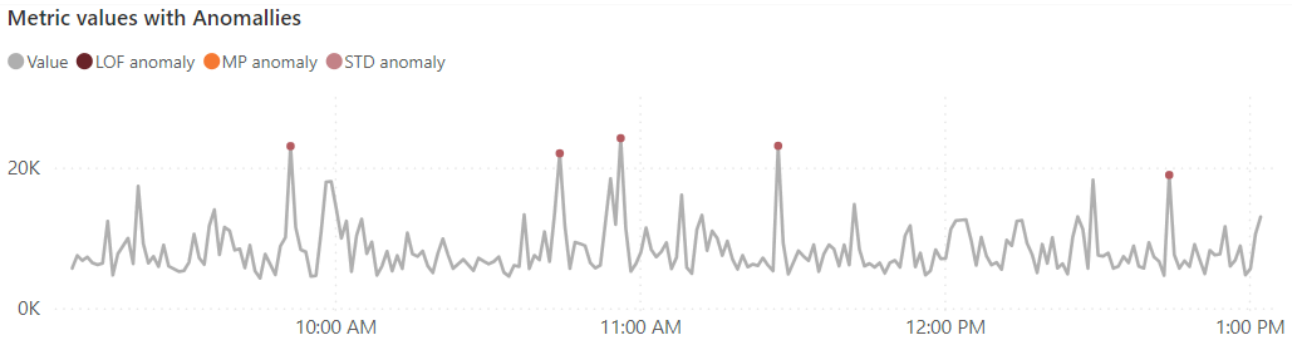

Рис 3.1.1 Приклад аномалія за стандартною оцінкою

Кількість таких аномалій по всім метрикам може бути зібрана в новий часовий ряд, на якому значна кількість аномалій може свідчити про помітний збій чи краш системи. (рис. 3.1.2) Тобто таку характеристику вже можливо

агрегувати по системі, що досі було неможливим для повністю різних метрик. Та підкреслимо незначні витрати часу для виявлення аномалій. Так медіанний час складає 0.0046 секунд на метрику, одна необхідність диференціювати і додатково перевіряти стаціонарність призводить до більш довгих розрахунків і більшого середнього часу у 0.1 секунди. (Середня кількість точок у ряді сягає 2500). Та нагадаємо, що розрахунок залежатиме від множнику z-score, рівню значущості для теста на стаціонарність та ліміт диференціювання, тобто ще три гіперпараметра.

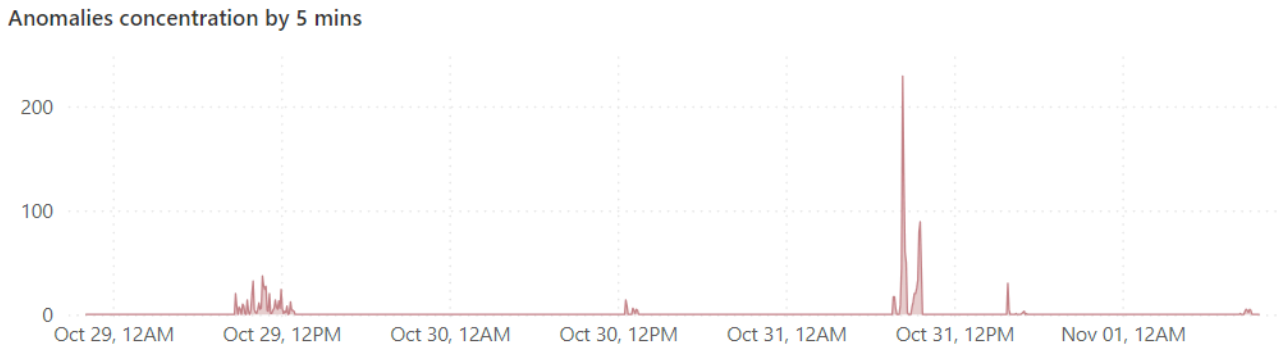

Рис 3.1.2 Концентрація аномалія за стандартною оцінкою у інфраструктурних метриках

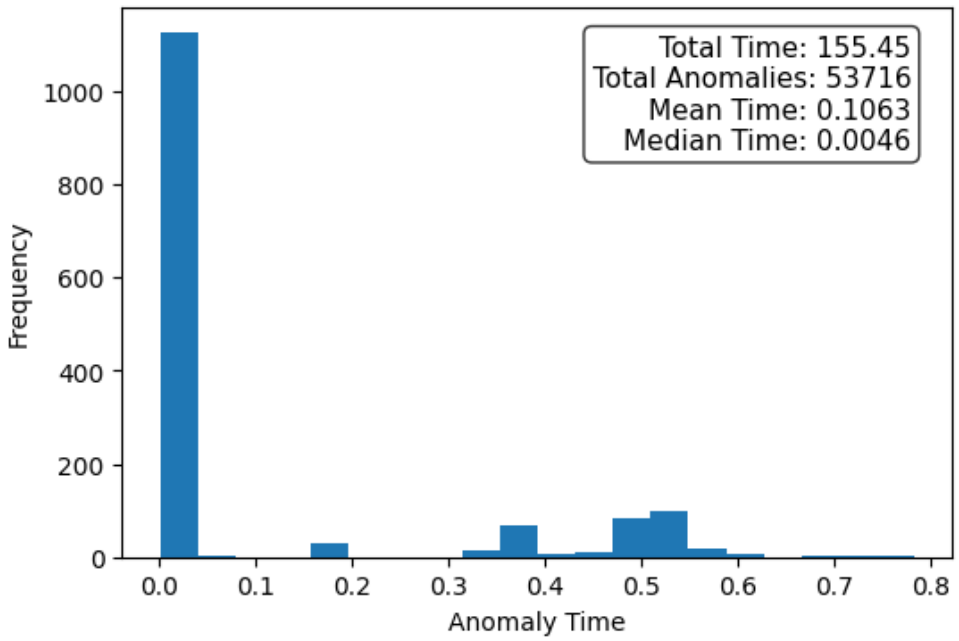

Рис 3.1.3 Розподіл затрат часу на виявлення аномалій

3.2 LOF

Іншим методом для пошуку аномалій був обраний Local Outlier Factor (LOF) для аналізу аномалій через його ефективність, швидкість та відносно низькі вимоги до обчислювальних ресурсів. Ці характеристики добре відповідають обмеженням поточної роботи, яка передбачає обробку часових рядів із різними характеристиками. Також LOF є одним із найпопулярніших методів unsupervised learning для виявлення аномалій. [12]

Ідея LOF полягає в тому, що метод враховує локальну щільність навколо кожної точки та порівнює її з середньою щільністю у сусідньому середовищі. Якщо щільність точки значно менша, ніж щільність її сусідства, вона вважається аномальною. Детальніше LOF обчислює локальний фактор аномальності (outlier factor), використовуючи відстань до k найближчих сусідів. Цей параметр ($n\_neighbors$) задає розмірність локального середовища і є гіперпараметром. Іншим гіперпараметром є рівень контамінації, який визначає частку очікуваних аномалій, що впливає на вибір порогу для класифікації.

$$Relative\ density\ of\ X = \frac{Density\ of\ X}{Average\ density\ of\ all\ data\ points\ in\ the\ neighborhood}$$

Слід зазначити, що метод досить чутливий до описаних гіперпараметрів і демонструє зростання обчислювальної складності на великих наборах даних. Для покращення результатів LOF часто поєднують з іншими методами аналізу. [11]

В цій роботі, на відміну від z-score, LOF генерує більш “шумні” дані щодо аномалій (рис 3.2.1), що не відміняє потенційні інсайти з цих даних.

Та LOF має деякі переваги перед z-score:

1. значна частина аномалій за стандартний відхилення будуть виявлені LOF
2. LOF здатний виявити аномалії у “період спокою” (рис 3.2.3)
3. LOF менше під впливом всієї виборки за своєю суттю

Але необхідність тонкого налаштування може приводити до хибних аномалій (рис 3.2.2), які потім і перетворюються на зайвий шум (рис. 3.2.1), це може бути з-за використання рівня контамінації, бо він примушує обирати хоч якісь аномалії. Можливим покращення може бути комбінація з вже відомими класами рядів та позбавлення сезонних та трендових компонентів, якщо можливо.

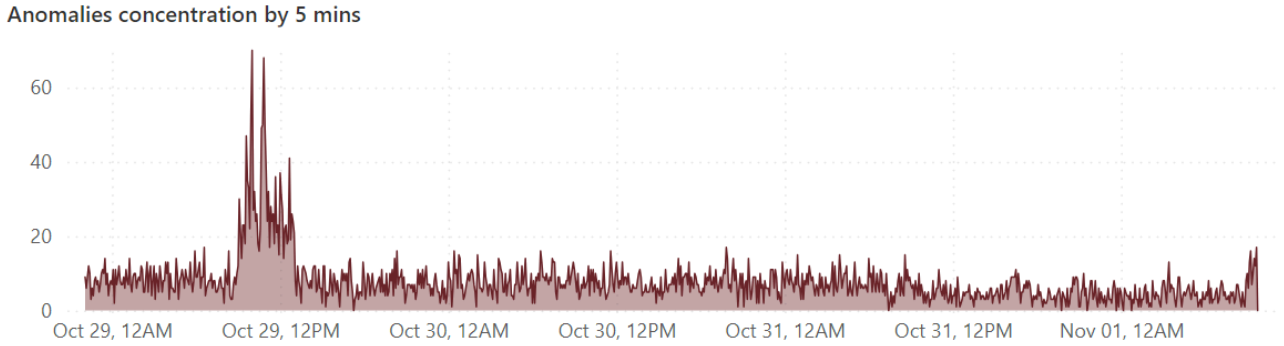

Рис 3.2.1 Концентрація аномалія за LOF у інфраструктурних метриках

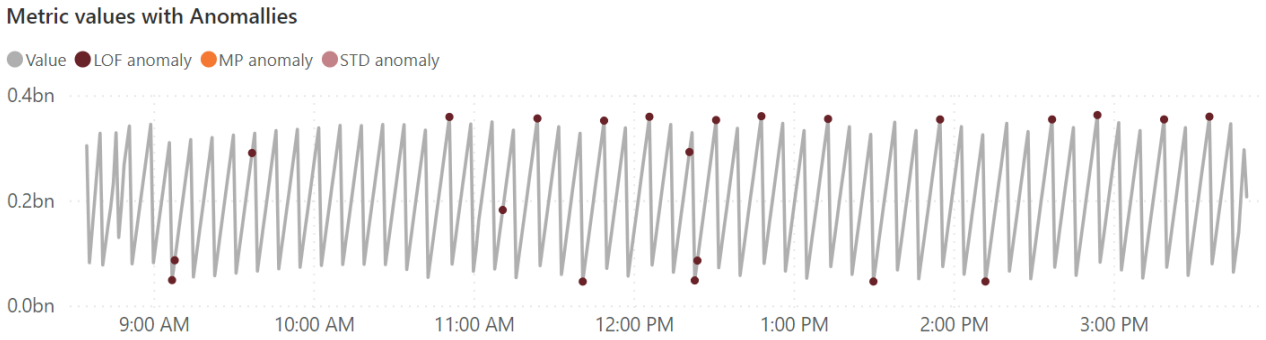

Рис 3.2.2 Невдале виділення аномалій за LOF

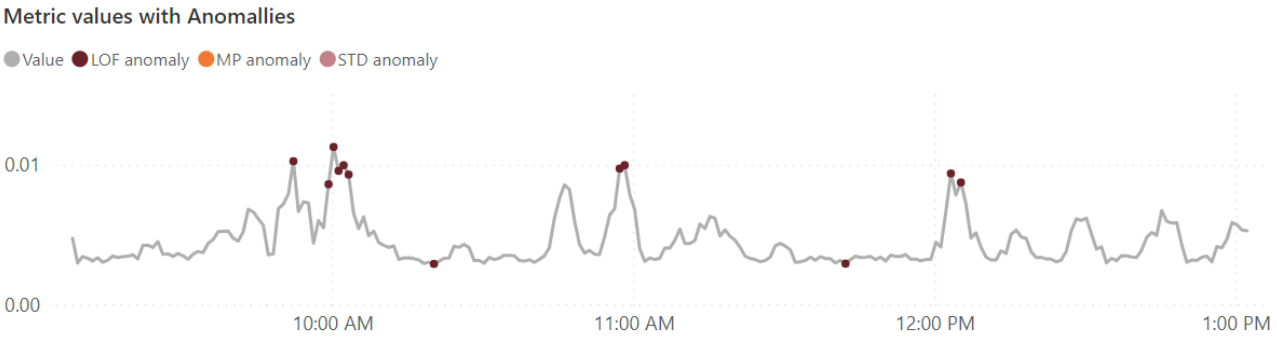

Рис 3.2.3 Досить вдале виділення аномалій за LOF

3.3 Matrix Profile

3.3.1 Визначення

Matrix Profile — це універсальний і потужний інструмент для аналізу часових рядів, який забезпечує автоматичне виявлення важливих патернів, аномалій, і повторюваних шаблонів.

Matrix Profile був запропонований дослідниками з Університету Каліфорнії в Ріверсайді у 2015-2016 роках і є досить молодим. Основною метою розробки була необхідність створити універсальний інструмент для аналізу часових рядів, який би поєднував швидкість, гнучкість і простоту реалізації. Стаття "Matrix Profile I: All Pairs Similarity Joins for Time Series" [13] стала основою для широкого використання цього методу в задачах аналізу часових рядів.

Метод ґрунтується на обчисленні *z-нормалізованих евклідових відстаней* між усіма підпослідовностями фіксованої довжини у часовому ряді. Цей процес включає:

1. Визначення підпослідовностей – у часовому ряді формуються всі можливі підпослідовності довжини m.
2. Обчислення евклідових відстаней – кожна підпослідовність порівнюється з усіма іншими, і обчислюється відстань між ними.
3. Збереження мінімальних відстаней – для кожної підпослідовності зберігається найменша відстань до іншої підпослідовності, що утворює Matrix Profile.

Основні властивості матричного профілю:

- Це вектор, в якому кожен елемент відображає найкоротшу відстань між певною підпослідовністю і її найближчим сусідом у часовому ряді.
- Забезпечує точні результати без хибно-позитивних чи хибно-негативних.

- Він є безпараметровим і не потребує налаштування порогових значень. Лише вибір одного гіперпараметру - довжини підпослідовності.
- Легко масштабується
- Обчислюється за $O(n^2)$, або за $O(n * log(n))$ разом з швидким перетворенням Фур'є (FFT). [13]

Головною причиною застосування матричного профілю - це ідеї мотивів та діскордів.

Мотив — це підпослідовність часових рядів, яка часто повторюється або має високу схожість з іншими підпослідовностями в даних. Мотиви є корисними для виявлення шаблонів, що регулярно з'являються, та для аналізу структурних особливостей часових рядів. У матричному профілі мотиви визначаються як пари позицій із найменшими значеннями профілю, що відповідають найближчим сусідам підпослідовностей у часових рядах. Тобто це можуть бути мінімуми у профілі чи малі значення.

Дискорд — це підпослідовність часових рядів, яка найбільше відрізняється від інших підпослідовностей. Іншими словами, дискорд — це аномалія або рідкісна подія, яка не має схожих сусідів. У матричному профілі дискорд визначається як позиція з найбільшим значенням профілю, що вказує на підпослідовність із найбільшою відстанню до свого найближчого сусіда. Тобто це максимуми у профілі, чи великі значення. [14]

3.3.2 Побудова

Побудова матричного профілю зовсім не потребує поперердьої обробки часового ряду аде вона сильно залежить від обраної довжини вікна (рис 3.3.1 та 3.3.2) (підпослідовності), що є новим гіперпараметром. Також присутній схований гіперпараметр - це вибір способу відокремлення мотивів та дискордів. В роботі мотиви та аномалії обиралися за перцентилями.

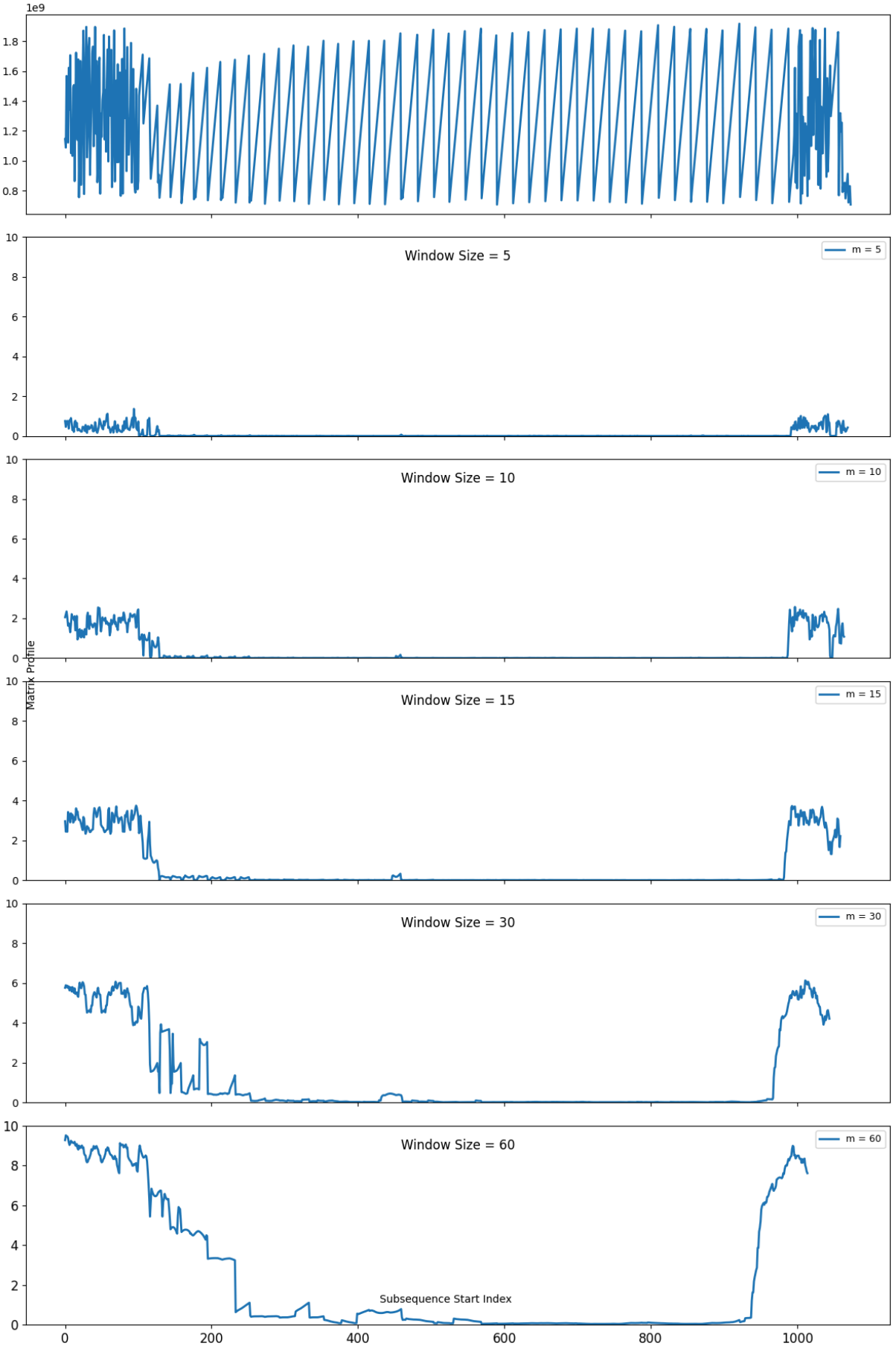

Рис 3.3.1 Матричні профілі JVM heap utilization

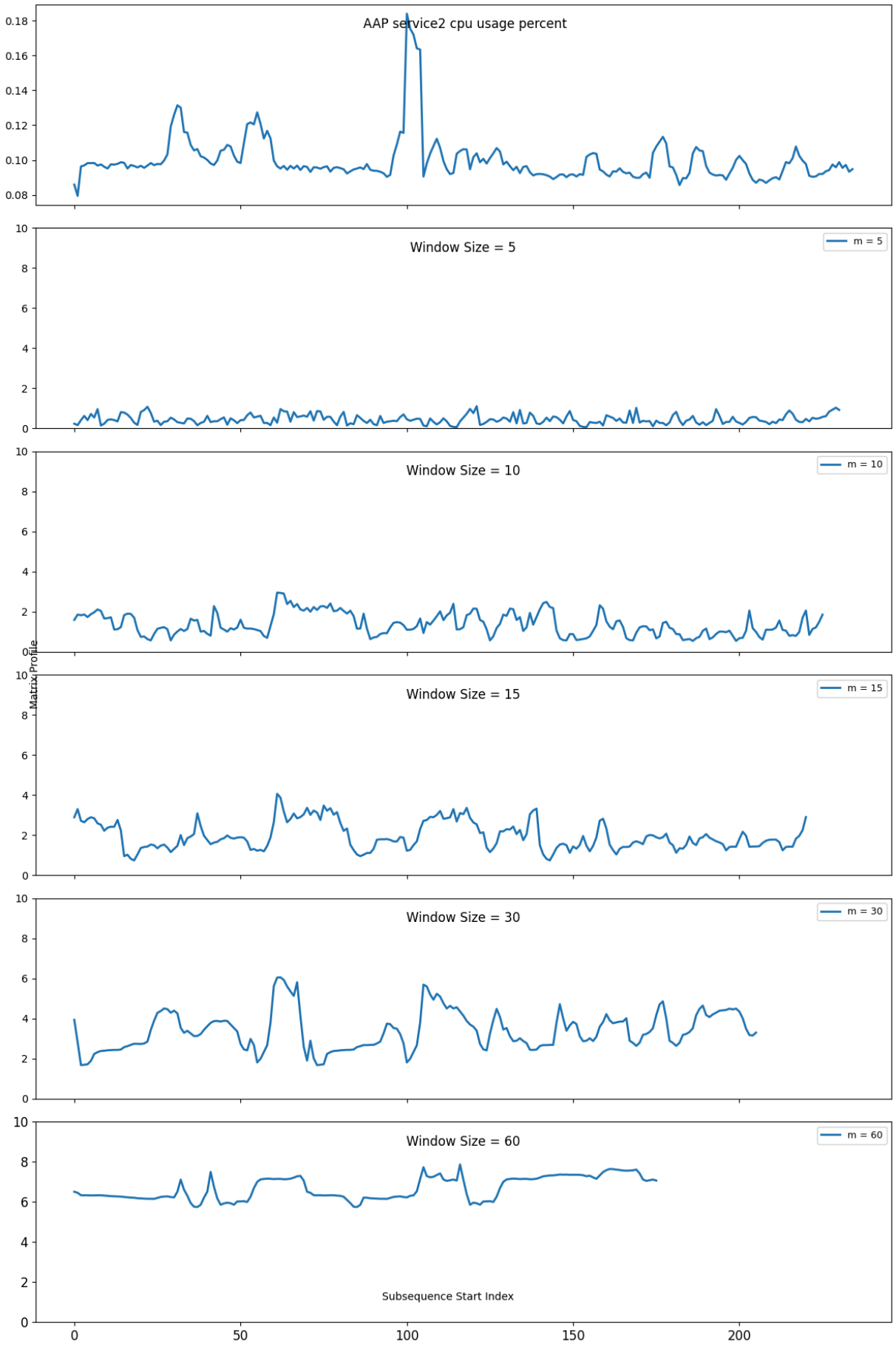

Рис 3.3.2 Матричні профілі CPU usage

3.3.3 Результати для одного ряду

Оцінюючи результати матричного профілю для однієї метрики, слід зазначити, що:

1. Дискорди та патерни можуть концентруватися в одному вікні і такі групи точок часто можна об’єднати у єдиний випадок (рис. 3.3.3, 3.3.4, 3.3.4) як “аномальну зону” ряду.
2. Визначення дискордів та мотивів відбувається однаково на тому самому профайлі, тобто матричний профіль виконує дві важливі задачі одночасно без додаткових витрат ресурсів.
3. На відміну від попередніх методів, матричний профіль здатен визнати аномалією частини зі спокоєм метрики, якщо це їй не властиво (рис. 3.3.3)
4. Матричний профіль відслідковує зміни у поведінці метрики навіть якщо ці зміни можуть залишитися непомітними для LOF чи станлартного відхилення (рис. 3.3.5), що робить метод помітно кориснішим.
5. Порушення патерну у ряді також може потрапити до аномалій (рис. 3.3.6)

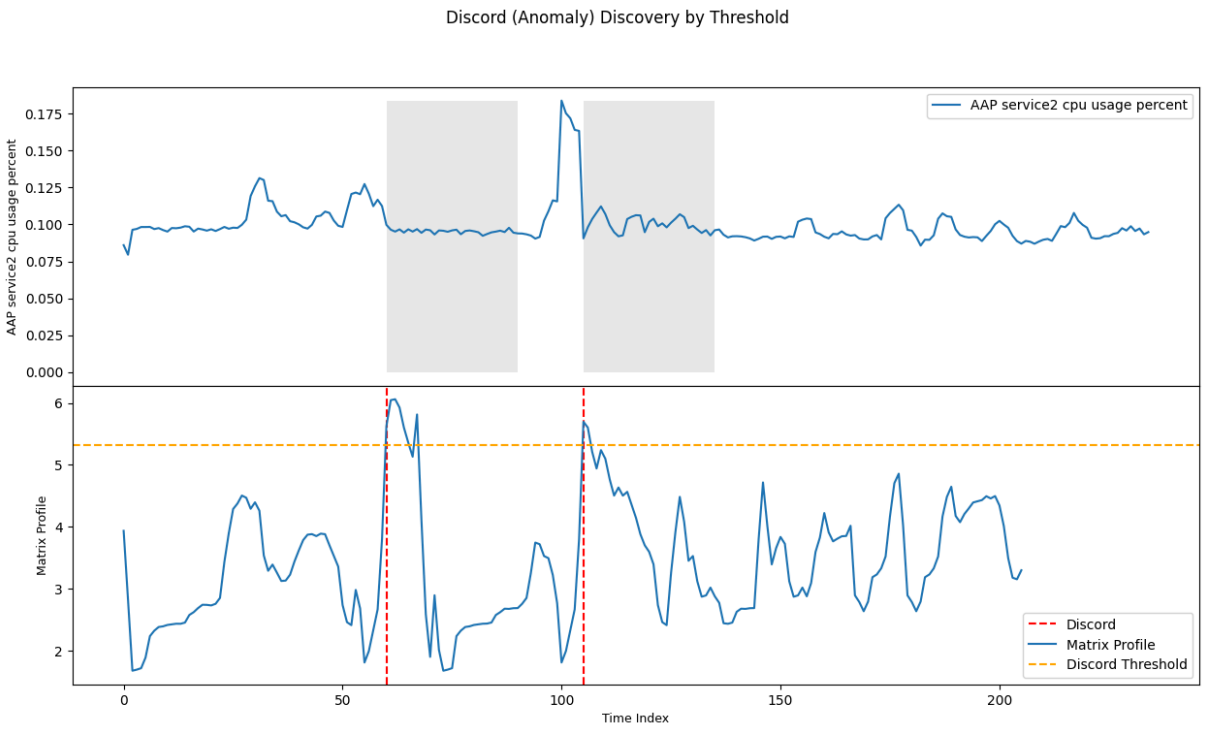

Рис 3.3.3 Дискорди (аномалії) у CPU usage

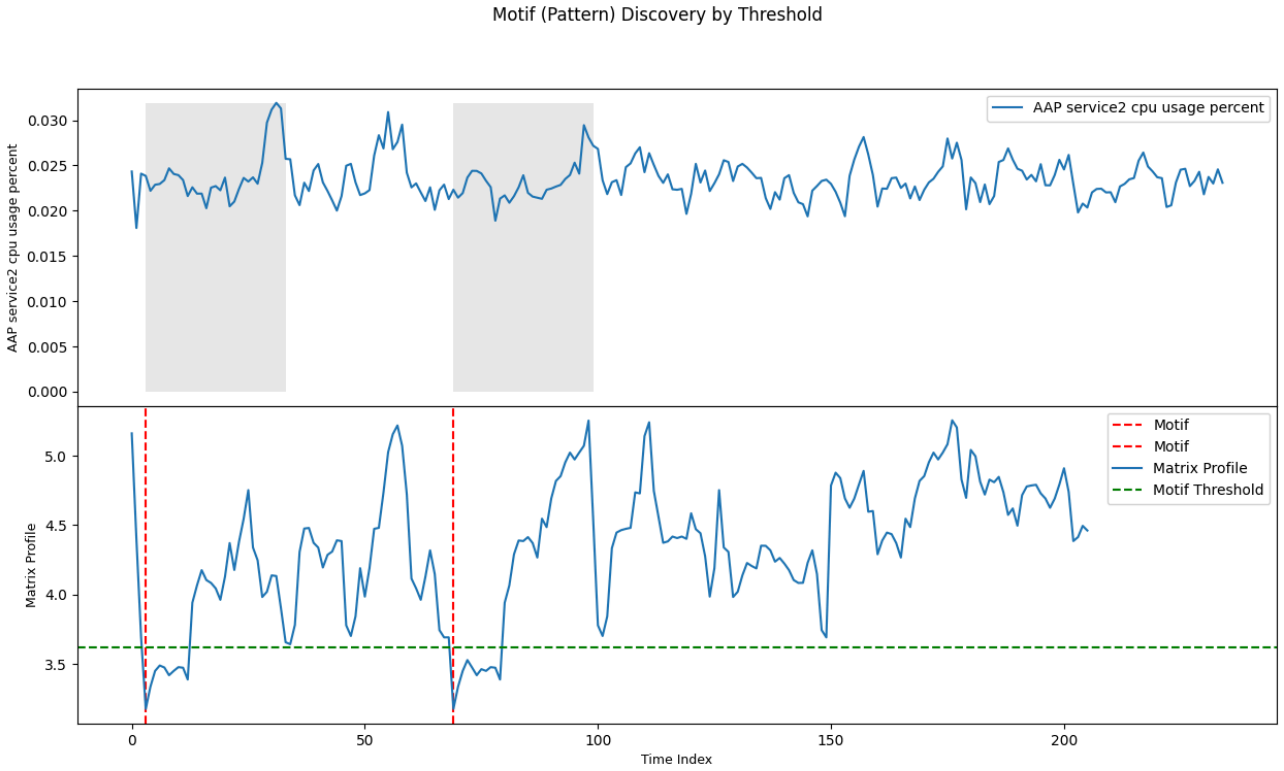

Рис 3.3.4 Мотиви (патерни) у CPU usage

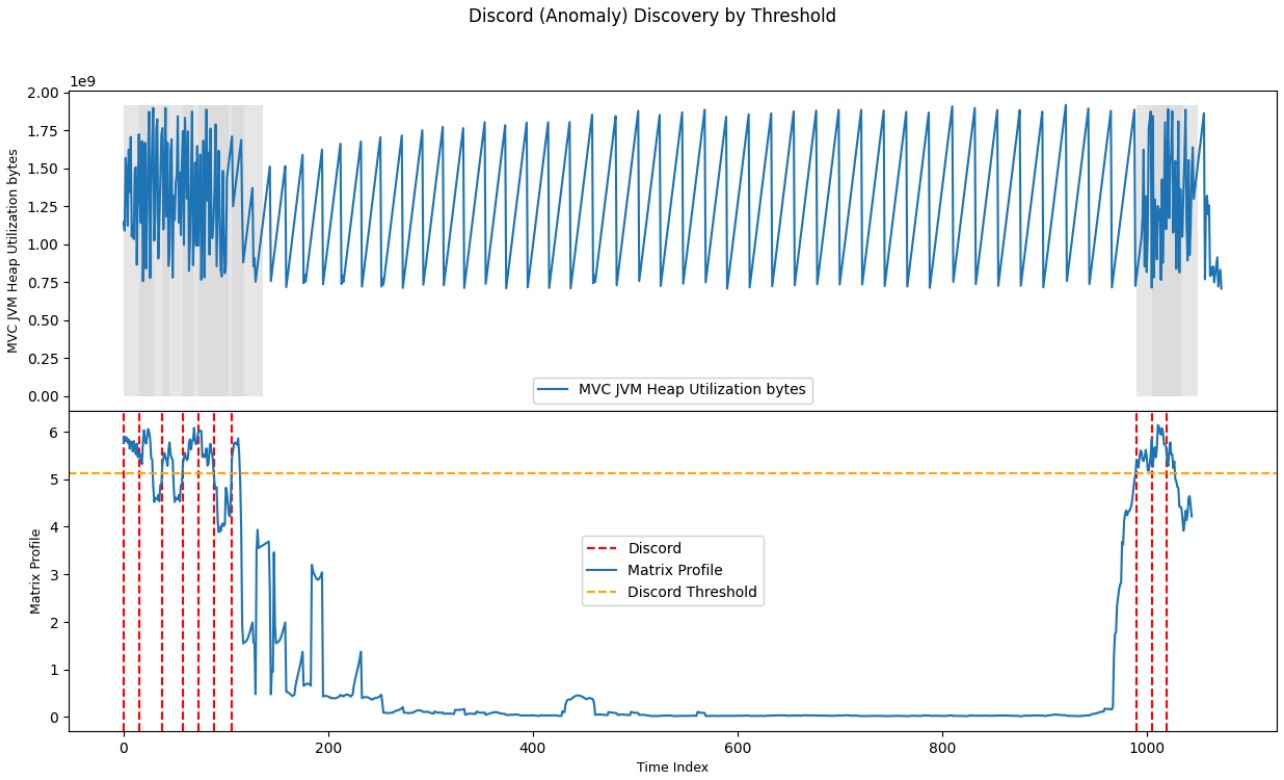

Рис 3.3.5 Дискорди (аномалії) у JVM heap utilization

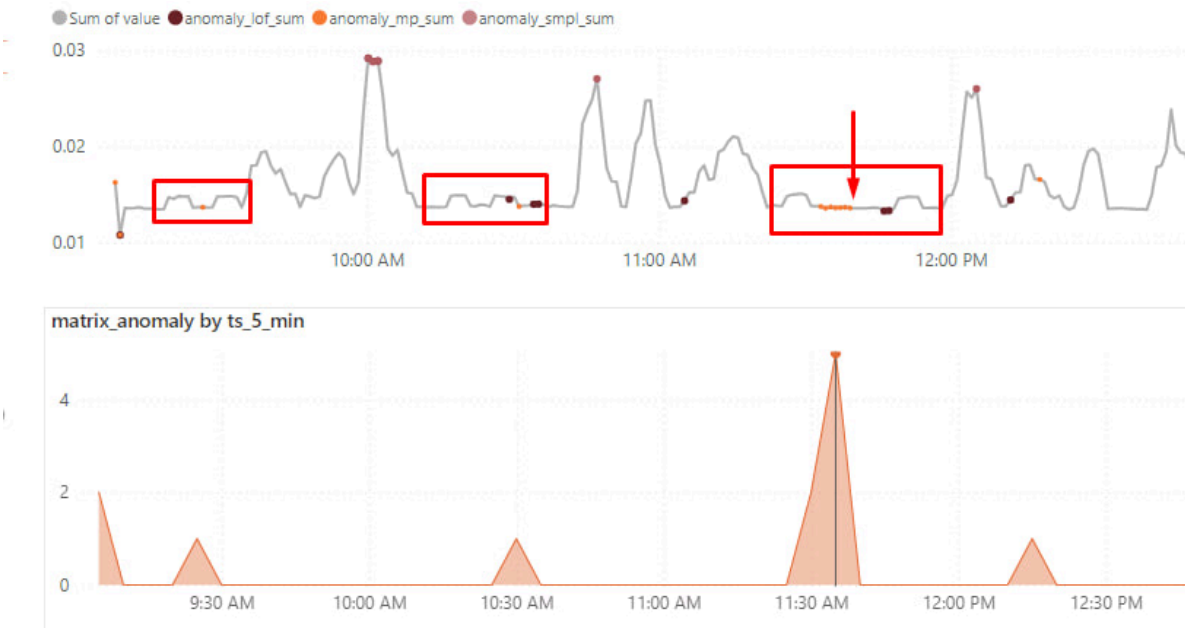

Рис 3.3.6 Порушення паттерну визначилися як аномалія

Інша концепція, яка випливає з матричного профілю — ланцюги часових рядів (Time Series Chains). Ці ланцюги є впорядкованими наборами підпослідовностей, де кожна підпослідовність схожа на попередню, але з можливістю еволюції чи зміни напряму. Основна ідея полягає у виявленні зв'язків між подібними підпослідовностями, що можуть відображати розвиток системи або зміну стану в часі. Ланцюги відрізняються від мотивів, оскільки їх елементи можуть дрейфувати чи змінюватись, тоді як мотиви відображають стабільні повторювані шаблони (рис 3.3.7). [15]

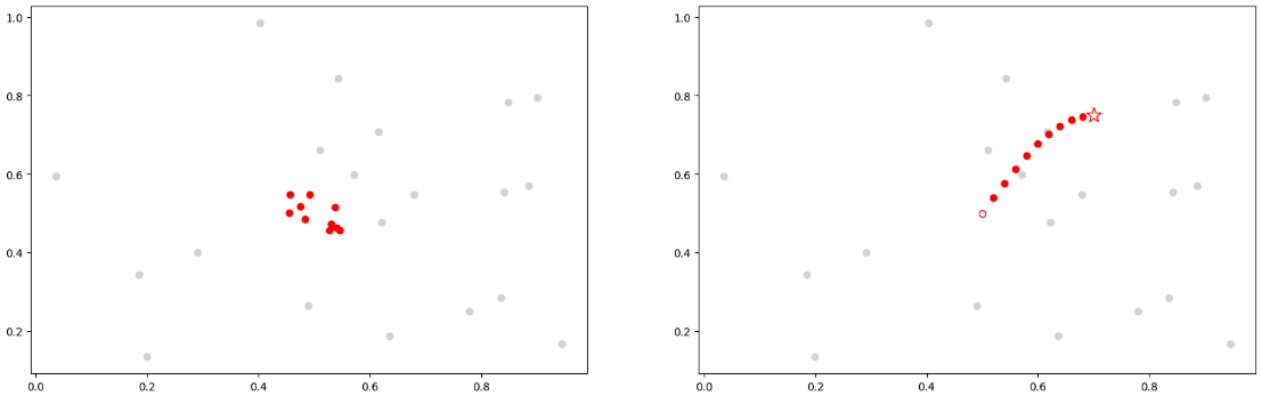

Рис 3.3.7 Візуалізація підпослідовностей часового ряду як точок у високовимірному просторі. Ліворуч: Мотив часового ряду можна розглядати як

сукупність точок, які наближаються до платонівського ідеалу, представленого тут як перехрестя Праворуч: На противагу цьому, ланцюг часового ряду можна розглядати як еволюцію точок у просторі. Тут перехрестя представляє першу ланку ланцюга, якір.[15]

В наявних даних, ланцюгами часових рядів можу слугувати коливання використання процесорів (рис 3.3.8) чи цикли у роботі JVM (рис 3.3.9). Також такі довгі ланцюги можуть слугувати показниками циклічності в даних, що вже відомо про дані JVM. Зміни в поведінці ланцюгів, або їх припинення може бути показником змін чи аномалій у процесах.

Іншим застосування може бути виявлення точок змін у часових рядів (рис 3.3.10), що цілком зрозуміло випливає з побудови метода. Такі потенційні точки будуть мати більше значення профілю і будуть схожі на діскорди.

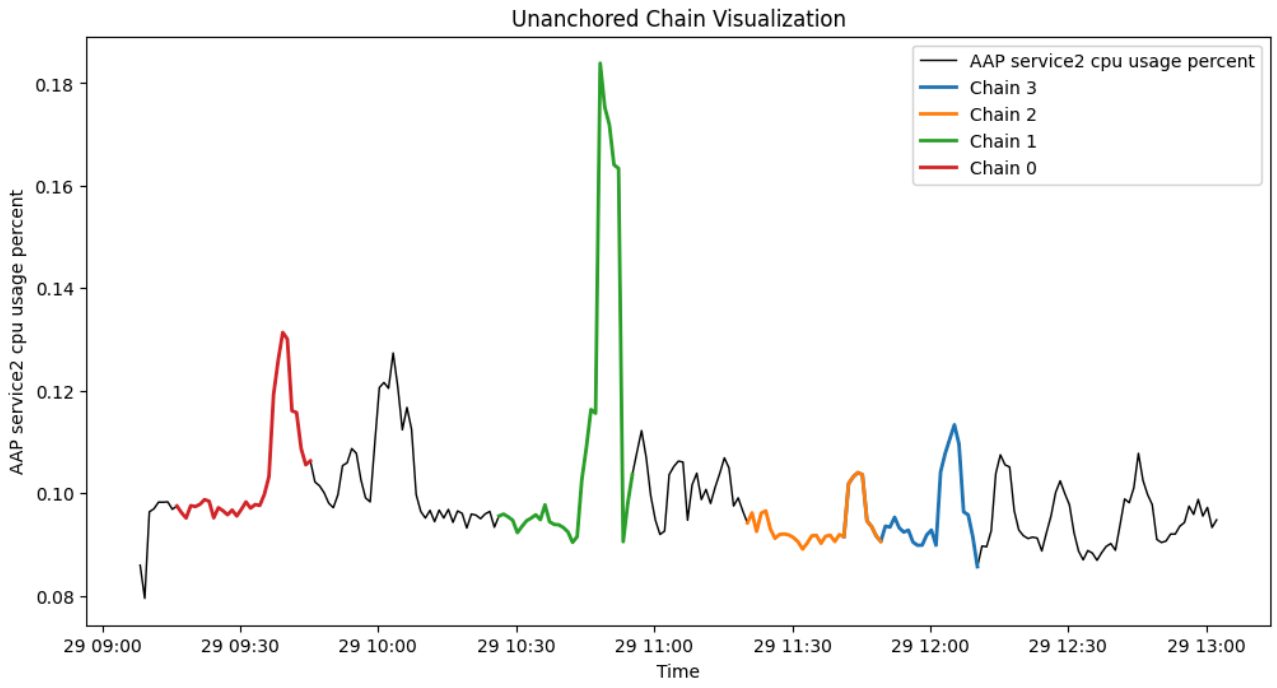

Рис 3.3.8 Ланцюг часових рядів у роботі CPU

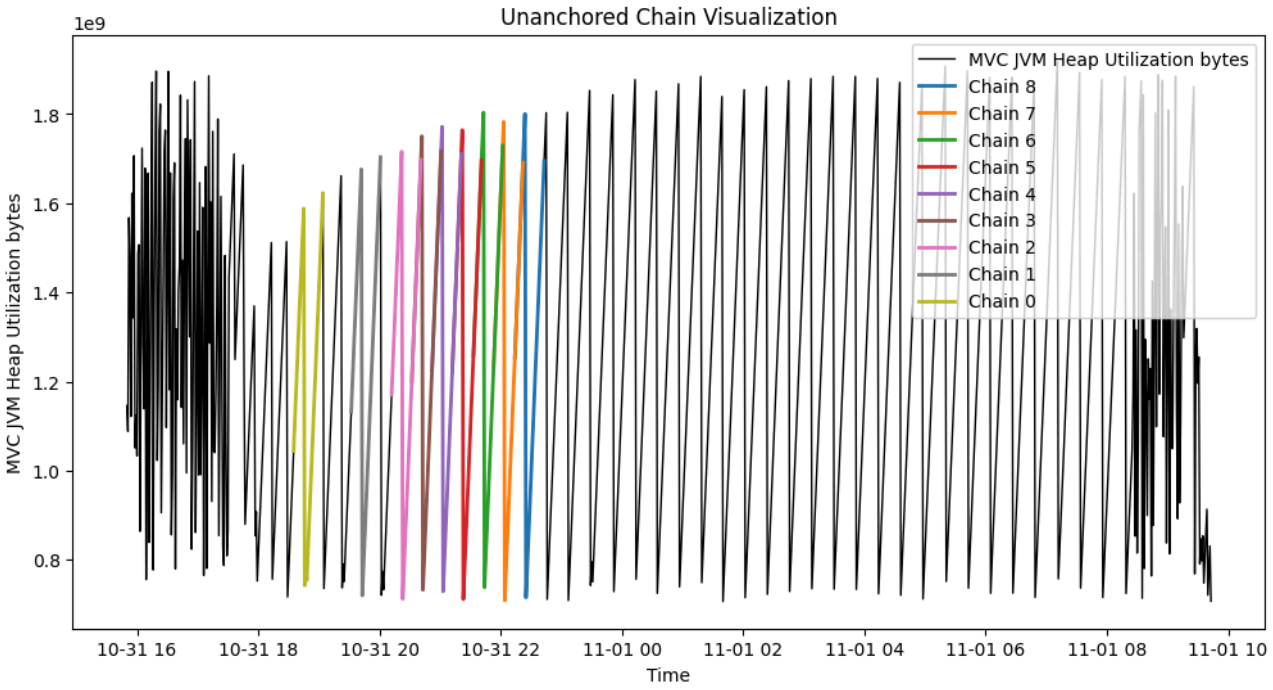

Рис 3.3.9 Ланцюг часових рядів у роботі JVM, що повторює циклічність

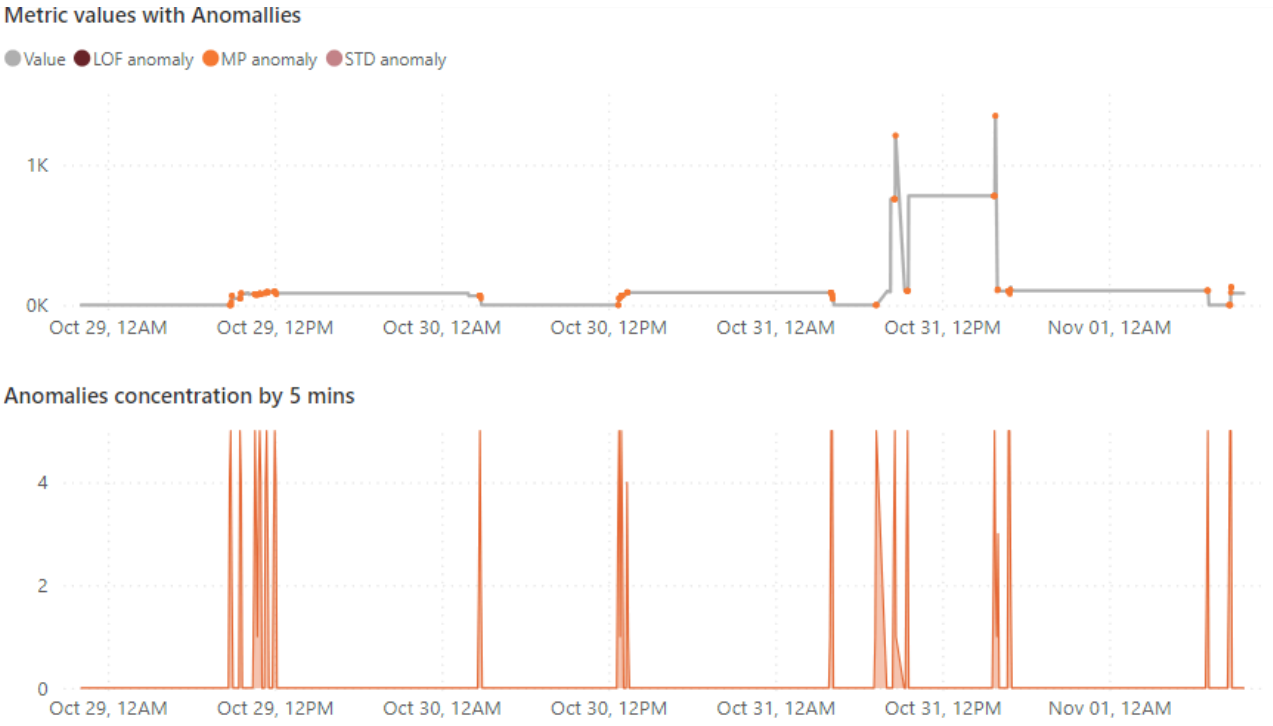

Рис 3.3.10 Виявлення точок змін у метриках

3.3.4 Результати для системи

Кількість аномалій, виявлених матричним профілем, так само можемо агрегувати по всім метрикам і звести до часового ряду аномалій у системі, на якому значна кількість аномалій може свідчити про помітний збій чи краш системи (рис. 3.3.11). Більш того, результати матричного профілю виявили додаткові 3-4 моменти концентрації аномалій, що не були виявлені попередніми методами (або не були чітко відокремлені).

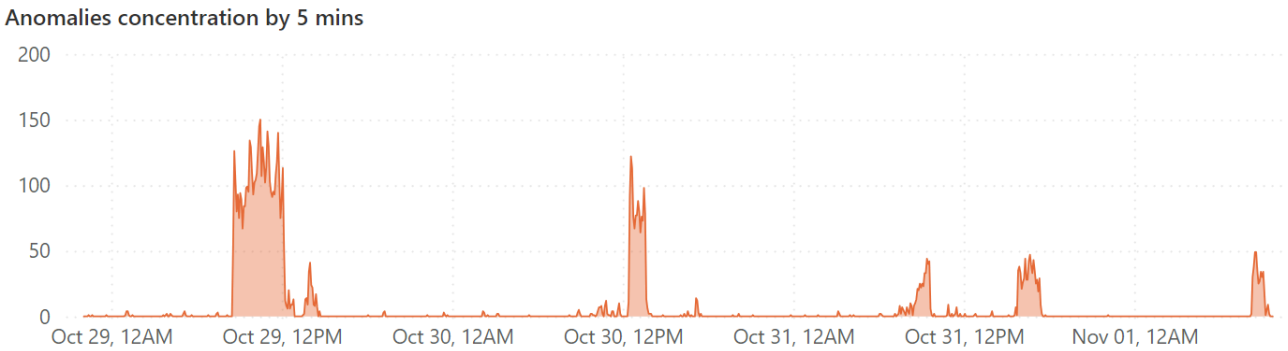

Рис 3.3.11 Виявлення точок змін у метриках

Також виділимо швидкість роботи методу на нашій виборці. Виявилось, що цей метод був найшвидшим за загальним часом.

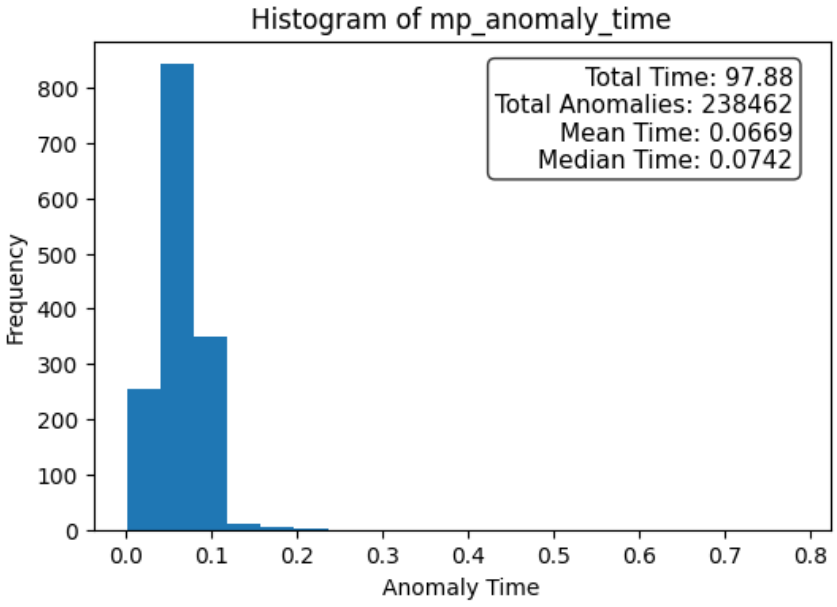

Рис 3.3.12 Затрати часу на пошук аномалій методом матричного профілю

3.4 Комбінування аномалій

Зберемо всі отримані аномалії разом для аналізу їх поведінки (рис. 3.4.1). Їх ідеї перетинаються, але в деяких аспектах кожна може мати перевагу, тому є сенс розглядати і кожний метод окремо і, можливо, всі разом.

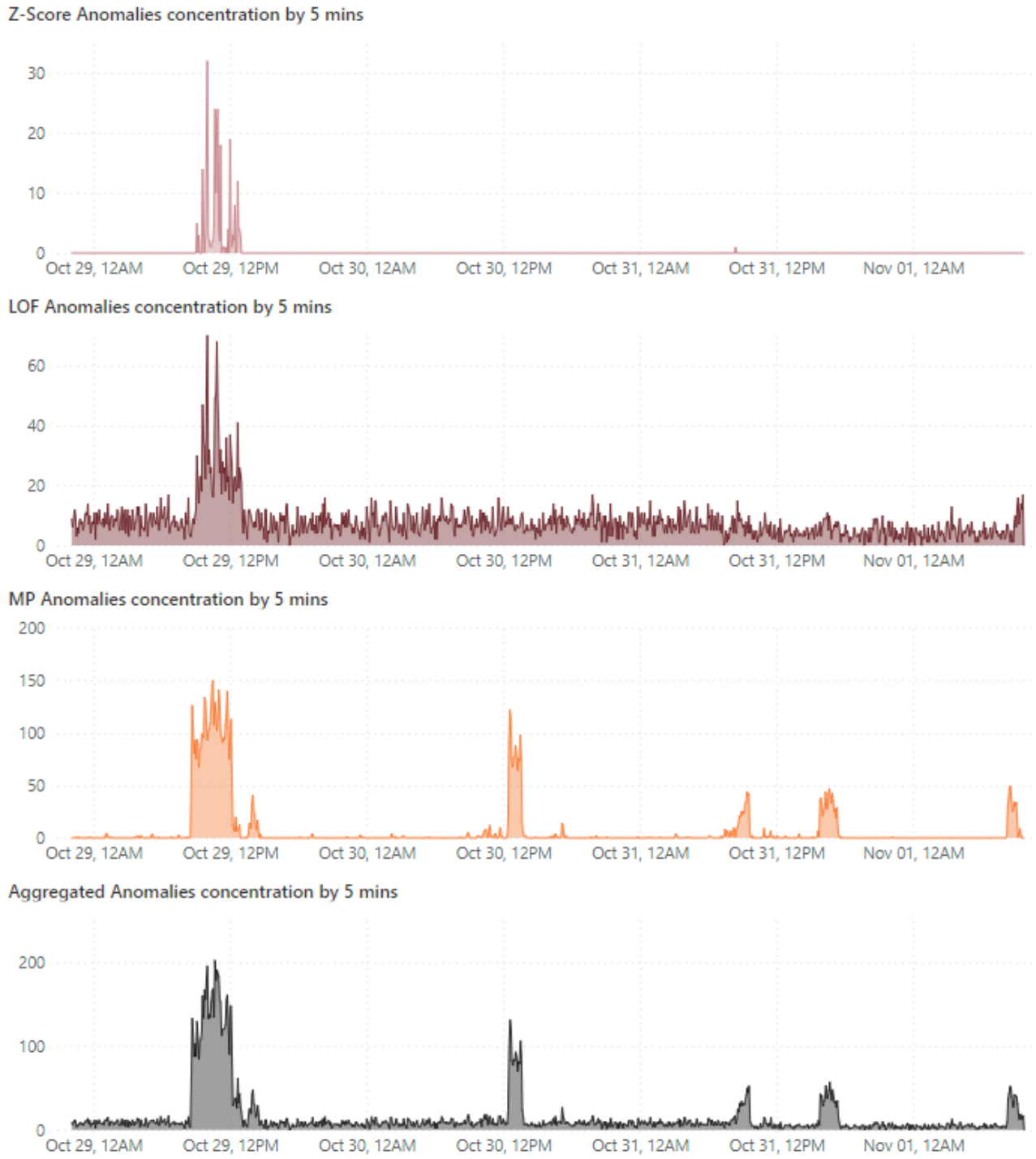

Рис 3.4.1 Порівняння концентрацій аномалій різних підходів

Але треба бути обережнішими з комбінацією, бо в отриманих результатах сильно домінує матричний профіль, тому слід бути обережними з агрегацією. Та зазначимо, що в кожному окремому випадку, аномалії можуть розподіляються незалежно і нести незалежні інсайти (рис. 3.4.2), так можуть і співпадати (рис. 3.4.1).

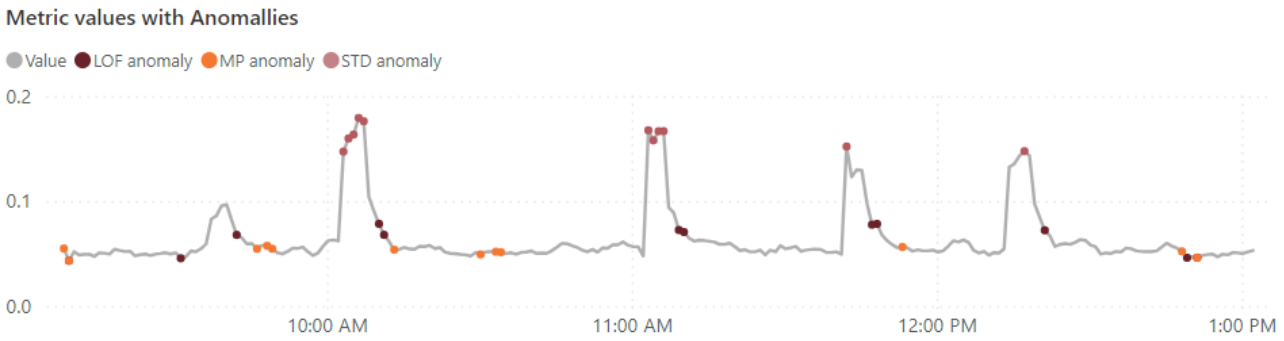

Рис 3.4.2 Різні аномалії в одній метриці

Також порівняємо часові витрати на пошук аномалій (рис. 3.4.3). Досить рівномірно витрачає час Матричний профіль і результаті він виявився найшвидшим в сумі і за середнім часом. Але z-score має значну перевагу в медіаному часі, бо насправді він є найшвидшим на рівні більшості метрик, затримки в його роботі спричинені необхідністю іноді проводити диференціювання. Найдовшим методом виявився LOF. Підсумовуючи, за більшістю ознак, найвигіднішим за можливостями і часом буде метод матричного профілю, але при дуже сильних обмеженнях у часі і ресурсах слід обирати метод z-score без диференціювання.

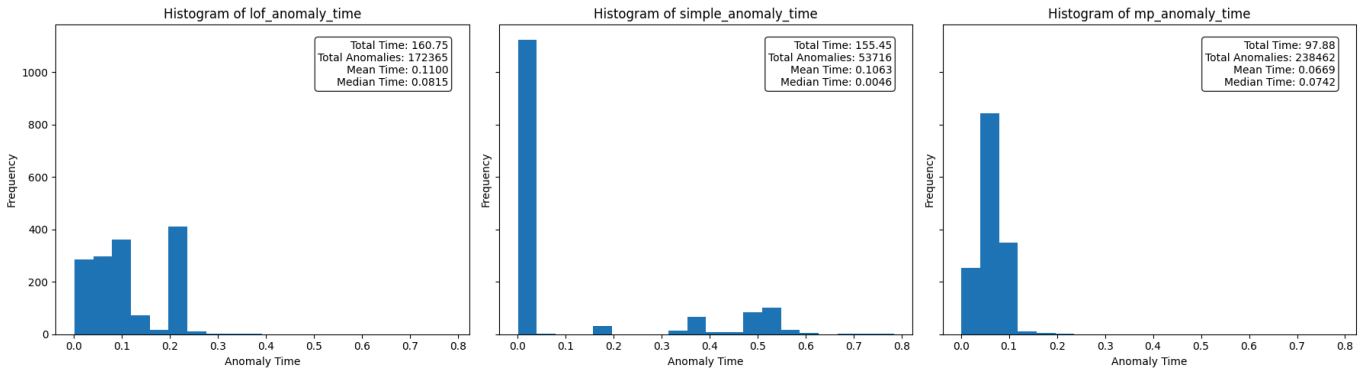

Рис 3.4.3 Розподіли витрат часу на пошук аномалій

ВИСНОВКИ

Під час виконання роботи були розглянуті наявні моніторингові системи, зібрані дані однієї з таких. Побудована автоматична класифікація системних метрик за типами часових рядів та автоматичне виявлення аномалій за допомогою методів z-score, LOF та Matrix Profile. Комбінація отриманих даних і методів дозволяє оцінювати як стан всієї системи так і стан конкретної метрики.

Повертаючись до головної проблеми, а саме – неможливість ручної оцінки метрик інформаційних систем з-за надвеликої їх кількості (яка може сягати сотен тисяч унікальних метрик). Можемо стверджувати, що:

1. Автоматична класифікація метрик дозволяє зводити аналіз системи до розгляду одного датасету, розміри якого будуть відповідати кількості самих метрик та обсягу метаданих цих метрик. Наприклад, швидке виявлення метрик, стан і поведінка яких не відповідає більшості у відповідній групі метрик (рис. 2.5.2), тощо. Тобто такий обсяг даних піддається в тому числі і ручному аналізу.
2. Автоматичне виявлення аномалій і зведення їх до одного часового ряду - концентрації аномалій у системі, дозволяє аналізувати всю систему з її різноманітними метриками целіком, виявляти моменти системних збоїв, структурні зміни тощо. А використання патернів замість аномалій у матричному профілі може призвести до визначення “стандартної” поведінки як системи, так і кожної метрики окремо. Це також колосально зменшує обсяг даних для подальшого моніторингу.

Більш виділивши методи виявлення аномалій (стандартизована оцінка, Local Outlier Factor, Matrix Profile), зазначимо, що їх використання разом може бути корисним, але затратним. До цього ж метод матричного профілю виявився найшвидшим і найбільш універсальним для ідентифікації аномалій та патернів у часових рядах, що робить його придатним для більш широкого застосування і

наводить на рекомендацію його використання загалом, та особливо у разі обмежень у ресурсах і часі.

Однак, незважаючи на ефективність розроблених методів, можливі покращення. Так тонке налаштування гіперпараметрів - це все ж таки найслабше місце роботи, бо більшість методів досить чутливі до них, тому вплив і ручне налаштування поки що є необхідним. Іншим покращенням може бути використання наявних методів разом з іншими підходами, або ж їх заміна на більш досконалі тощо. Також покращенням може бути використання паралелізації для економії часу.

Підсумовуючи і незважаючи на зазначені недоліки, можемо стверджувати, що розглянутий підхід і рішення є універсальними та життєздатними для впровадження в реальних інформаційних системах. Він робить можливим моніторинг за всіми тисячами метрик, може забезпечити автоматизацію процесу аналізу великих обсягів даних, покращити виявлення аномалій і патернів в системі, але буде потребувати тонкого налаштування аналітичної системи та побудови додаткових систем оповіщення та візуалізації.

СПИСОК ЛІТЕРАТУРИ

1. An Introduction to Metrics, Monitoring, and Alerting. / DigitalOcean, 2017 URL:https://www.digitalocean.com/community/tutorials/an-introduction-to-metrics-monitoring-and-alerting (дата звернення: 13.11.2024)
2. What is Prometheus. / Prometheus, 2023 URL: https://prometheus.io/docs/introduction/overview/ (дата звернення: 20.11.2024)
3. Time series data and analysis. / InfluxDB, URL: https://www.influxdata.com/what-is-time-series-data/#time-series-data (дата звернення: 20.11.2024)
4. Kwiatkowski, D., Phillips, P. C. B., Schmidt, P., & Shin, Y. Testing the null hypothesis of stationarity against the alternative of a unit root. / Journal of Econometrics. 1992, розд. 54 с. 159–178.
5. Augmented Dickey Fuller Test (ADF Test) – Must Read Guide / machine learning +, 2019, URL:https://www.machinelearningplus.com/time-series/augmented-dickey-fuller-test/ (дата звернення: 20.11.2024)
6. STAT 510 Applied Time Series Analysis, 6.1 The Periodogram URL: https://online.stat.psu.edu/stat510/lesson/6/6.1 (дата звернення: 11.11.2024)
7. Hyndman, R.J., & Athanasopoulos, G. Forecasting: principles and practice, 3rd edition, 2021, Melbourne, Australia. URL: OTexts.com/fpp3 (дата звернення: 01.11.2024)
8. Cleveland, R. B., Cleveland, W. S., McRae, J. E., & Terpenning, I. J. STL: A seasonal-trend decomposition procedure based on loess. Journal of Official Statistics, 1990, 6(1), 3–33. URL: http://bit.ly/stl1990 (дата звернення: 10.11.2024)
9. Seasonal-Trend decomposition using LOESS (STL). / statsmodels, 2024 URL:https://www.statsmodels.org/dev/examples/notebooks/generated/stl_decomposition (дата звернення: 10.11.2024)

10. Nóbrega S., The Ultimate Guide to Finding Outliers in Your Time-Series Data (Part 1)./ Towards Data Science 2024
URL:https://towardsdatascience.com/the-ultimate-guide-to-finding-outliers-in-your-time-series-data-part-1-1bf81e09ade4 (дата звернення: 19.11.2024)
11. Vijay Kotu, Bala Deshpande, Data Science (Second Edition), 2019, розд. 13.4 Local Outlier Factor,
URL:https://www.sciencedirect.com/book/9780128147610/data-science (дата звернення: 19.11.2024)
12. Comparing anomaly detection algorithms for outlier detection on toy datasets / scikit-learn, 2024,
URL:https://scikit-learn.org/1.5/auto_examples/miscellaneous/plot_anomaly_comparison.html#sphx-glr-auto-examples-miscellaneous-plot-anomaly-comparison-py (дата звернення: 19.11.2024)
13. Yeh C.M., Zhu Y., Ulanova L., Begum N., Ding Y., Dau H.A., Silva D.F., Mueen A., Keogh E., Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets. / IEEE ICDM, 2016.
URL:https://www.cs.ucr.edu/%7Eeamonn/PID4481997_extend_Matrix%20Profile_I.pdf (дата звернення: 21.11.2024)
14. The Matrix Profile, Stumpy documentation,
URL:https://stumpy.readthedocs.io/en/latest/Tutorial_The_Matrix_Profile.html (дата звернення: 21.11.2024)
15. Zhu Y., Imamura M., Nikovski D., Keogh E., Matrix Profile VII: Time Series Chains: A New Primitive for Time Series Data Mining. / ICDM, 2017
URL: http://www.cs.ucr.edu/%7Eeamonn/chains_ICDM.pdf (дата звернення: 21.11.2024)
16. A. Blázquez-García, A. Conde, Mori U., Lozano J., A review on outlier/anomaly detection in time series data, 2020
URL: https://arxiv.org/pdf/2002.04236 (дата звернення: 21.11.2024)

17. Devi, K.L., Valli, S. Time series-based workload prediction using the statistical hybrid model for the cloud environment. / 2023, розд. Computing 105, с. 353–374. URL:https://doi.org/10.1007/s00607-022-01129-7
18. Machine Learning Glossary. / Google, 2024
URL: https://developers.google.com/machine-learning/glossary?hl=en

ДОДАТОК А

КОД ПРОГРАМИ КЛАСИФІКАЦІЇ ТА ПОШУКУ АНОМАЛІЙ

```
import os
import json
import re
import hashlib
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
from pathlib import Path
from statsmodels.tsa.stattools import adfuller, acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.seasonal import STL
from scipy.signal import periodogram, find_peaks, detrend

#### Script params
NEED_LOAD = False
ADD_LOF = True
# Classification hyperparameters
DATA_LACK_QTY=50
CONSTANT_LVL=0.99
CONF_A=0.05
TREND_LVL=0.3
SEASONAL_STRENGTH_THRESHOLD=0.5
# Anomalies
Z_THRESHOLD = 3
MP_WINDOW = 10
MP_PERCENTILE = 5
DIFF_LIMIT = 3
# LOF hyperparameters
LOF_N_NEIGHBORS = 20
```

```
LOF_CONTAMINATION = 0.05

#### Data loading and Classification


def generate_metric_id(metadata):

   metadata_string = "|".join(f"{key}={metadata[key]}" for key in
sorted(metadata))

   metric_id = hashlib.md5(metadata_string.encode()).hexdigest()

   return metric_id


def estimate_seasonal_period(data, max_lag=None):

   """Estimates the seasonal period of a time series using Periodogram and
ACF"""

   data = data.dropna()

   data_detrended = detrend(data)

   frequencies, spectrum = periodogram(data_detrended)

   # Exclude zero frequency

   frequencies = frequencies[1:]

   spectrum = spectrum[1:]

   if len(spectrum) == 0:

       return None

   peaks, _ = find_peaks(spectrum)

   if len(peaks) == 0:

       return None


   # Dominant frequency

   dominant_peak = peaks[np.argmax(spectrum[peaks])]

   dominant_freq = frequencies[dominant_peak]

   if dominant_freq == 0:

       return None


   period_from_periodogram = int(round(1 / dominant_freq))


   # Additional check using ACF

   if max_lag is None:

       max_lag = min(len(data) // 2, 100)
```

```
    autocorr = acf(data_detrended, nlags=max_lag)
    peaks_acf, _ = find_peaks(autocorr[1:])  # exclude lag 0

    if len(peaks_acf) == 0:
        return None
    period_from_acf = peaks_acf[0] + 1
    if (abs(period_from_periodogram - period_from_acf) <= 2) or
(period_from_periodogram in peaks_acf):
        return period_from_periodogram
    else:
        return None

def classify_time_series(df, data_lack_qty=DATA_LACK_QTY,
constant_lvl=CONSTANT_LVL, conf_a=CONF_A, trend_lvl=TREND_LVL,

seasonal_strength_threshold=SEASONAL_STRENGTH_THRESHOLD):
    """Classifies a time series for stationarity, trend, and seasonality"""
    start_time = time.perf_counter()
    data = df['value'].dropna()
    mode_qty = data.value_counts().iloc[0]
    sample_size = data.size
    constant_score = mode_qty / sample_size
    is_data_lack = sample_size < data_lack_qty
    is_constant = constant_score > constant_lvl
    res = {
        'constant_score': constant_score,
        'is_data_lack': is_data_lack,
        'is_constant': is_constant,
        'mode': data.mode().iloc[0],
        'sample_size': sample_size
    }

    is_stationary = False
    is_seasonal = False
    is_trend = False
    seasonal_period = None
```

```
    seasonal_strength = None
    trend_strength = None
    adf_statistic = None
    adf_pvalue = None

    if not is_data_lack and not is_constant:
        # Remove trend and seasonality for stationarity test
        detrended_data = detrend(data)
        adf_result = adfuller(detrended_data, regression='ct')
        adf_statistic, adf_pvalue = adf_result[0], adf_result[1]
        is_stationary = adf_pvalue < conf_a
        # Estimate seasonal period
        seasonal_period = estimate_seasonal_period(data)
        if seasonal_period is not None and sample_size >= 2 * seasonal_period:
            try:
                # Apply STL decomposition
                stl = STL(data, period=seasonal_period, robust=True)
                result = stl.fit()
                seasonal = result.seasonal
                trend = result.trend
                resid = result.resid

                # Estimate seasonal strength
                resid_var = np.var(resid) + 1e-8  # prevent division by zero
                seasonal_var = np.var(seasonal)
                seasonal_strength = max(0, 1 - (resid_var / (resid_var +
seasonal_var)))
                is_seasonal = seasonal_strength > seasonal_strength_threshold

                # Estimate trend strength
                trend_var = np.var(trend)
                trend_strength = max(0, 1 - (resid_var / (resid_var +
trend_var)))
                is_trend = trend_strength > trend_lvl
```

```
                # Re-check stationarity on residuals
                adf_resid = adfuller(resid)
                is_stationary = adf_resid[1] < conf_a
            except Exception as e:
                res['error'] = str(e)
    else:
        is_stationary = False
        is_seasonal = False
        is_trend = False

    end_time = time.perf_counter()
    execution_time = end_time - start_time

    res.update({
        'is_stationary': is_stationary,
        'is_seasonal': is_seasonal,
        'is_trend': is_trend,
        'adf_statistic': adf_statistic,
        'adf_pvalue': adf_pvalue,
        'seasonal_period': seasonal_period,
        'seasonal_strength': seasonal_strength,
        'trend_strength': trend_strength,
        'execution_time_seconds': execution_time
    })
    return res

if NEED_LOAD:
    directory_path = './datasets'
    time_series_data = []
    metric_metadata = {}
    additional_cols = set()
    for batch_index, subfolder in enumerate(Path(directory_path).iterdir()):
        # if batch_index == 3:
            print(f"Processing batch: {subfolder.name} (Batch index:
{batch_index})")
```

```
            for file_path in subfolder.glob("*.json"):
                file_name = file_path.stem
                suffix = re.sub(r'\d+', '', file_name)
                suffix = re.sub(r'_+', '_', suffix)
                print("  Processing file:", file_name)
                print("  Suffix:", suffix)
                # Load JSON data
                with open(file_path, 'r') as f:
                    data = json.load(f)
                for metric_dict in data.get('data', [{}]):
                    for metric_name, metric_data in metric_dict.items():
                        print('Processing...', metric_name)
                        status = metric_data['status']
                        status = 'No data' if status == 'success' and
len(metric_data['data']['result']) == 0 else status


                        if status == 'success' or status == 'No data':
                            for result in metric_data['data']['result']:
                                values = result['values']
                                df = pd.DataFrame(values, columns=['timestamp',
'value'])
                                df['timestamp'] = pd.to_datetime(df['timestamp'],
unit='s')
                                df['value'] = df['value'].astype(float)
                                  metric_md = {}
                                metric_md['metric_name'] = metric_name
                                metric_md['batch'] = batch_index
                                for key, value in result['metric'].items():
                                    metric_md[key] = value
                                    additional_cols.add(key)
                                metric_id = generate_metric_id(metric_md)
                                metric_md['id'] = metric_id
                                for key, value in
classify_time_series(df).items():
                                    metric_md[key] = value
```

```
                                if metric_id in metric_metadata:
                                    if metric_metadata[metric_id]['sample_size']
< metric_md['sample_size']:
                                        metric_metadata[metric_id] = metric_md
                                else:
                                    metric_metadata[metric_id] = metric_md
                                df['metric_name'] = metric_name
                                df['mark'] = suffix
                                df['batch'] = batch_index
                                df['metric_id'] = metric_id
                                time_series_data.append(df)
                            else:
                                print(metric_dict,'\n')


    # Concatenate all data into a single DataFrame
    ts_df = pd.concat(time_series_data)
    ts_df.to_csv('timeseries_data.csv', index=False)
    meta_df = pd.DataFrame.from_dict(metric_metadata, orient='index')
    meta_df.to_csv('metric_metadata.csv', index=True)
    meta_df_filtered = meta_df.drop(columns=list(additional_cols))
    meta_df_filtered = meta_df_filtered[meta_df_filtered['is_data_lack'] ==
False]
    meta_df_filtered.to_csv('clear_metric_metadata.csv', index=True)
else:
    ts_df = pd.read_csv('timeseries_data.csv')
    meta_df = pd.read_csv('metric_metadata.csv', index_col=0)
    meta_df_filtered = pd.read_csv('clear_metric_metadata.csv', index_col=0)


#### Simple anomaly detection
meta_df[['is_data_lack', 'is_constant', 'is_stationary', 'is_seasonal',
'is_trend']].value_counts()


from scipy.stats import zscore
def detect_anomalies_simple(df, metadata, z_threshold=Z_THRESHOLD,
max_diff=DIFF_LIMIT, adf_pvalue_threshold=CONF_A, print_err=False):
    """Detect anomalies in a time series based on its classification, with
differencing for nonstationary data."""
```

```
    start_time = time.perf_counter()
    data = df.copy()
    data = data.sort_values('timestamp').reset_index()
    is_constant = metadata.get('is_constant', False)
    is_stationary = metadata.get('is_stationary', False)
    is_seasonal = metadata.get('is_seasonal', False)
    is_trend = metadata.get('is_trend', False)
    seasonal_period = metadata.get('seasonal_period', None)
    data['anomaly_simple'] = False
    def is_stationary_adf(series, pvalue_threshold=0.05):
        result = adfuller(series.dropna())
        return result[1] < pvalue_threshold
    try:
        if is_constant:
            constant_value = metadata.get('mode', data['value'].mode().iloc[0])
            data['anomaly_simple'] = data['value'] != constant_value
        elif is_seasonal:
            # Use STL
            if seasonal_period is None:
                seasonal_period = estimate_seasonal_period(data['value'])
                if seasonal_period is None:
                    raise ValueError(f"Seasonal error")
            stl = STL(data['value'], period=seasonal_period, robust=True)
            result = stl.fit()
            resid = result.resid
            resid_zscore = zscore(resid)
            data['resid_anomaly'] = resid_zscore.abs() > z_threshold
            trend_diff = result.trend.diff().abs()
            trend_threshold = trend_diff.mean() + z_threshold * trend_diff.std()
            data['trend_anomaly'] = trend_diff > trend_threshold
            data['anomaly_simple'] = data['resid_anomaly'] |
data['trend_anomaly']
            data = data.drop(columns=['resid_anomaly', 'trend_anomaly'])
        elif not is_stationary:
            diff_order = 0
```

```
            diff_series = data['value']
            while not is_stationary_adf(diff_series) and diff_order < max_diff:
                diff_series = diff_series.diff().dropna()
                diff_order += 1
            if diff_order > 0 and is_stationary_adf(diff_series):
                data['value_diff'] = diff_series
                data['z_score'] = zscore(data['value_diff'])
                data.loc[data['z_score'].abs() > z_threshold, 'anomaly_simple'] =
True
                data = data.drop(columns=['z_score', 'value_diff'])
            else:
                if print_err:
                    raise ValueError(f"Non-stationarity could not be resolved for
the given time series after {max_diff} differences.")
        else:
            data['z_score'] = zscore(data['value'].fillna(data['value'].mean()))
            data['anomaly_simple'] = data['z_score'].abs() > z_threshold
            data = data.drop(columns=['z_score'])
    except Exception as e:
        if print_err:
            print(f"Error in anomaly detection: {e} | id: {metadata.get('id',
-1)}")
        data['z_score'] = zscore(data['value'].fillna(data['value'].mean()))
        data['anomaly_simple'] = data['z_score'].abs() > z_threshold
        data = data.drop(columns=['z_score'])
    end_time = time.perf_counter()
    execution_time = end_time - start_time
    data.set_index('index', inplace=True)
    return data, execution_time


#### Matrix Profile
import stumpy
def detect_anomalies_matrix_profile(df, metadata, m=MP_WINDOW,
mp_percentile=MP_PERCENTILE):
    """Detect anomalies and patterns in a time series using Matrix Profile"""
    start_time = time.perf_counter()
```

```
    data = df.copy()
    data = data.sort_values('timestamp')
    data = data.reset_index()
    values = data['value'].values
    if len(values) < m * 2:
        data['anomaly_mp'] = 'none'
        execution_time = time.perf_counter() - start_time
        return data, execution_time
    mp = stumpy.stump(values, m, ignore_trivial=True)
    # Matrix Profile values and indices
    mp_values = mp[:, 0]
    mp_indices = mp[:, 1]
    discord_threshold_value = np.percentile(mp_values, 100-mp_percentile)  # Top
5% as discords
    discord_positions = np.where(mp_values > discord_threshold_value)[0]
    # For motifs
    motif_threshold_value = np.percentile(mp_values, mp_percentile)  # Bottom 5%
as motifs
    motif_positions = np.where(mp_values < motif_threshold_value)[0]
    data['anomaly_mp'] = 'none'
    data.loc[discord_positions, 'anomaly_mp'] = 'discord'
    data.loc[motif_positions, 'anomaly_mp'] = 'motif'
    end_time = time.perf_counter()
    execution_time = end_time - start_time
    data.set_index('index', inplace=True)
    return data, execution_time


from sklearn.neighbors import LocalOutlierFactor
def detect_anomalies_lof(df, metadata, n_neighbors=LOF_N_NEIGHBORS,
contamination=LOF_CONTAMINATION):
    """Detect anomalies in a time series using Local Outlier Factor (LOF)"""
    start_time = time.perf_counter()
    data = df.copy()
    data = data.sort_values('timestamp')
    data = data.reset_index()
```

```
    values = data['value'].values.reshape(-1, 1)
    if len(values) < n_neighbors:
        data['anomaly_lof'] = 'none'
        execution_time = time.perf_counter() - start_time
        data.set_index('index', inplace=True)
        return data, execution_time
    # Apply LOF
    lof = LocalOutlierFactor(n_neighbors=n_neighbors,
contamination=contamination)
    lof_labels = lof.fit_predict(values)
    lof_scores = -lof.negative_outlier_factor_

    # Identify anomalies
    anomaly_threshold = np.percentile(lof_scores, 100 * (1 - contamination))
    data['anomaly_lof'] = lof_scores > anomaly_threshold
    end_time = time.perf_counter()
    execution_time = end_time - start_time
    data.set_index('index', inplace=True)
    return data, execution_time


metric_metadata = meta_df.to_dict('index')
meta_filtered_dict = meta_df_filtered.to_dict('index')
len(metric_metadata.items())
from tqdm import tqdm
anomaly_detection_times = {}
for metric_id, metadata in tqdm(metric_metadata.items(), desc="Processing
metrics"):
    if not metadata['is_data_lack']:
        df_metric = ts_df[(ts_df['metric_id'] == metric_id) & (ts_df['batch'] ==
metadata['batch'])]
        # simple anomaly detection method
        df_simple_anomalies, simple_time = detect_anomalies_simple(df_metric,
metadata)
        ts_df.loc[df_simple_anomalies.index, 'anomaly_simple'] =
df_simple_anomalies['anomaly_simple']
        metric_metadata[metric_id]['simple_anomaly_time'] = simple_time
```

```
        # Matrix Profile anomaly

        df_mp_anomalies, mp_time = detect_anomalies_matrix_profile(df_metric,
metadata)

        ts_df.loc[df_mp_anomalies.index, 'anomaly_mp'] =
df_mp_anomalies['anomaly_mp']

        metric_metadata[metric_id]['mp_anomaly_time'] = mp_time

        # LOF anomaly

        if ADD_LOF:

            df_lof_anomalies, lof_time = detect_anomalies_lof(df_metric,
metadata)

            ts_df.loc[df_lof_anomalies.index, 'anomaly_lof'] =
df_lof_anomalies['anomaly_lof']

            metric_metadata[metric_id]['lof_anomaly_time'] = lof_time

            metric_metadata[metric_id]['num_lof_anomalies'] =
df_lof_anomalies['anomaly_lof'].sum()



        metric_metadata[metric_id]['num_simple_anomalies'] =
df_simple_anomalies['anomaly_simple'].sum()

        metric_metadata[metric_id]['num_mp_anomalies'] =
df_mp_anomalies['anomaly_mp'].value_counts().get('discord', 0)

# Save the updated data

ts_df.to_csv('timeseries_data_with_anomalies.csv', index=False)

meta_df = pd.DataFrame.from_dict(metric_metadata, orient='index')

meta_df.to_csv('metric_metadata_with_times.csv', index=True)

def plot_anomalies(df, metric_id):

    data = df[df['metric_id'] == metric_id]

    plt.figure(figsize=(12, 6))

    plt.plot(data['timestamp'], data['value'], label='Value')

    anomalies_simple = data[data['anomaly_simple'] == True]

    anomalies_mp = data[data['anomaly_mp'] == 'discord']

    plt.scatter(anomalies_simple['timestamp'], anomalies_simple['value'],
color='red', label='Simple Anomalies')

    plt.scatter(anomalies_mp['timestamp'], anomalies_mp['value'], color='orange',
label='Matrix Profile Discords')

    if ADD_LOF:

        anomalies_lof = data[data['anomaly_lof'] == True]

        plt.scatter(anomalies_lof['timestamp'], anomalies_lof['value'],
color='green', label='LOF Anomalies')

    plt.title(f"Anomalies for Metric ID: {metric_id}")
```

```
    plt.xlabel('Timestamp')
    plt.ylabel('Value')
    plt.legend()
    plt.show()
```

ДОДАТОК Б

КОД ПРОГРАМИ МАТРИЧНОГО ПРОФІЛЯ

```
import pandas as pd
ts_df = pd.read_csv('timeseries_data.csv')
meta_df = pd.read_csv('metric_metadata.csv', index_col=0)
meta_df.loc[:, ['metric_name', 'batch', 'host', 'profile', 'version', 'catalog',
'dev_unit', 'site', 'db', 'instance', 'namespace']].head()
METRIC_ID = '83f4a2b2853c58ffba69e309ed149d7c'
metric_name = 'cpu usage' # default

if METRIC_ID is None:
    ts_df = ts_df[ts_df['metric_name'] == 'AAP service2 cpu usage percent']
    meta_df = meta_df[meta_df['metric_name'] == 'AAP service2 cpu usage
percent'].dropna(axis=1, how='all')
else:
    metric_name = meta_df.loc[METRIC_ID].metric_name
    ts_df = ts_df[ts_df['metric_name'] == metric_name]
    meta_df = meta_df[meta_df['metric_name'] == metric_name].dropna(axis=1,
how='all')
import numpy as np
import matplotlib.pyplot as plt
import stumpy

filtered_df = ts_df[ts_df['metric_id'] == METRIC_ID]
filtered_df['timestamp'] = pd.to_datetime(filtered_df['timestamp'])
filtered_df.set_index('timestamp', inplace=True)

values = filtered_df['value'].values
window_sizes = [5, 10, 15, 30, 60]

fig, axs = plt.subplots(len(window_sizes) + 1, sharex=True, figsize=(12, 18))
fig.text(0.5, 0.04, 'Subsequence Start Index', ha='center')
fig.text(0.04, 0.5, 'Matrix Profile', va='center', rotation='vertical')

axs[0].plot(values, label=f"{metric_name}", linewidth=2)
axs[0].set_title(f" {metric_name}",  y=0.85)
```

```
for i, window_size in enumerate(window_sizes):
    mp = stumpy.stump(values, m=window_size)
    i = i + 1
    axs[i].plot(mp[:, 0], label=f"m = {window_size}", linewidth=2)
    axs[i].set_ylim(0, 10)
    axs[i].set_title(f"Window Size = {window_size}",  y=0.85)
    axs[i].legend(fontsize=9)

plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.tight_layout()
plt.show()

window_size = 30
window_shift = 0 # 1 or 0
from matplotlib.patches import Rectangle
from scipy.signal import argrelextrema

def filter_indices(indices, min_distance):
    filtered = []
    last_index = -min_distance
    for idx in indices:
        if idx - last_index >= min_distance:
            filtered.append(idx)
            last_index = idx
    return np.array(filtered)


mp = stumpy.stump(values, m=window_size).astype(np.float64)
mean_mp = np.mean(mp[:, 0])
std_mp = np.std(mp[:, 0])
motif_threshold = mean_mp - 1.5 * std_mp
discord_threshold = mean_mp + 2 * std_mp
motif_indices = np.where(mp[:, 0] < motif_threshold)[0]
discord_indices = np.where(mp[:, 0] > discord_threshold)[0]

motif_indices = filter_indices(motif_indices, window_size // 2)
discord_indices = filter_indices(discord_indices, window_size // 2)

print(f"Motif indices: {motif_indices}")
print(f"Discord indices: {discord_indices}")
```

```
fig, axs = plt.subplots(2, sharex=True, figsize=(15, 8), gridspec_kw={'hspace':
0})
plt.suptitle('Motif (Pattern) Discovery by Threshold', fontsize=12)

axs[0].plot(values, label=f"{metric_name}")
axs[0].set_ylabel(f"{metric_name}", fontsize=9)
for motif_idx in motif_indices:
   rect = Rectangle((motif_idx - window_shift * window_size // 2, 0), window_size,
max(values), facecolor='lightgrey', alpha=0.5)
   axs[0].add_patch(rect)
axs[0].legend()
axs[1].set_xlabel('Time Index', fontsize=9)
axs[1].set_ylabel('Matrix Profile', fontsize=9)
for motif_idx in motif_indices:
   axs[1].axvline(x=motif_idx, linestyle="dashed", color='red', label='Motif')
axs[1].plot(mp[:, 0], label="Matrix Profile")
axs[1].axhline(y=motif_threshold, color='green', linestyle="--", label="Motif
Threshold")
axs[1].legend()
plt.show()

fig, axs = plt.subplots(2, sharex=True, figsize=(15, 8), gridspec_kw={'hspace':
0})
plt.suptitle('Discord (Anomaly) Discovery by Threshold', fontsize=12)

axs[0].plot(values, label=f"{metric_name}")
axs[0].set_ylabel(f"{metric_name}", fontsize=9)
for discord_idx in discord_indices:
   rect = Rectangle((discord_idx - window_shift * window_size // 2, 0),
window_size, max(values), facecolor='lightgrey', alpha=0.5)
   axs[0].add_patch(rect)
axs[0].legend()
axs[1].set_xlabel('Time Index', fontsize=9)
axs[1].set_ylabel('Matrix Profile', fontsize=9)
for discord_idx in discord_indices:
   axs[1].axvline(x=discord_idx, linestyle="dashed", color='red', label='Discord')
axs[1].plot(mp[:, 0], label="Matrix Profile")
axs[1].axhline(y=discord_threshold, color='orange', linestyle="--", label="Discord
Threshold")
axs[1].legend()
handles, labels = plt.gca().get_legend_handles_labels()
by_label = dict(zip(labels, handles))
axs[1].legend(by_label.values(), by_label.keys())
```

```
plt.show()

values = filtered_df['value'].values.astype(np.float64)

mp = stumpy.stump(values, m=window_size)

all_chain_set, unanchored_chain = stumpy.allc(mp[:, 2], mp[:, 3])

plt.figure(figsize=(12, 6))
plt.plot(filtered_df.index, values, linewidth=1, color='black',
label=f"{metric_name}")
for i in range(unanchored_chain.shape[0] - 1, -1, -1):
   chain_start = unanchored_chain[i]
   y = values[chain_start:chain_start + window_size]
   x = filtered_df.index[chain_start:chain_start + window_size]
   plt.plot(x, y, linewidth=2, label=f"Chain {i}")

plt.legend()
plt.title("Unanchored Chain Visualization")
plt.xlabel("Time")
plt.ylabel(f"{metric_name}")
plt.show()
```