МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

БАЛЮРА БОГДАН ПАВЛОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д. т. н., професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

АЛГОРИТМИ ТА ПРОГРАМНІ ЗАСОБИ ВИЯВЛЕННЯ АНОМАЛІЙ У ДАНИХ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Петро НІКОЛЮК, професор кафедри
інформаційних технологій,
д-р фіз.-мат. наук, професор
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця – 2026

АНОТАЦІЯ

Балюра Б. П. Алгоритми та програмні засоби виявлення аномалій у даних з використанням штучного інтелекту. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2026.

У кваліфікаційній (бакалаврській) роботі досліджено теоретичні основи виявлення аномалій у табличних наборах даних та проведено порівняльний аналіз алгоритмів машинного навчання без учителя. Розроблено десктопний програмний засіб AnomalyAI на основі Python 3.10+ та Qt6, який реалізує чотири алгоритми детекції (Isolation Forest, LOF, One-Class SVM, PCA-автоенкодер), дев'ять типів інтерактивних графіків та модуль AI Copilot для природномовної інтерпретації аномалій із підтримкою офлайн-режиму.

Ключові слова: виявлення аномалій, машинне навчання, Isolation Forest, One-Class SVM, автоенкодер, великі мовні моделі, Python, Qt6.

55 ст. 24 рис., 15 табл., 1 дод., 45 джерел.

ABSTRACT

Baliura B. P. Algorithms and Software Tools for Anomaly Detection in Data Using Artificial Intelligence. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2026.

In the qualification (bachelor's) work the theoretical foundations of anomaly detection in tabular datasets are investigated and a comparative analysis of unsupervised machine learning algorithms is conducted. A desktop software tool AnomalyAI has been developed using Python 3.10+ and Qt6, implementing four detection algorithms (Isolation Forest, LOF, One-Class SVM, PCA-Autoencoder), nine types of interactive charts, and an AI Copilot module for natural language interpretation of anomalies with offline mode support.

Keywords: anomaly detection, machine learning, Isolation Forest, One-Class SVM, autoencoder, large language models, Python, Qt6.

55 p. 24 fig., 15 tabl., 1 append., 45 ref.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI — Artificial Intelligence — штучний інтелект

API — Application Programming Interface — інтерфейс програмування застосунків

AUC — Area Under the Curve — площа під кривою

AWS — Amazon Web Services — хмарна платформа Amazon

CSV — Comma-Separated Values — формат табличних даних, розділених комами

F1 — F1-score — гармонічне середнє точності та повноти

FN — False Negative — хибно негативний результат класифікації

FP — False Positive — хибно позитивний результат класифікації

FPR — False Positive Rate — частка хибно позитивних результатів

GPU — Graphics Processing Unit — графічний процесор

GUI — Graphical User Interface — графічний інтерфейс користувача

HTML — HyperText Markup Language — мова розмітки гіпертексту

HTTP — HyperText Transfer Protocol — протокол передачі гіпертексту

IF — Isolation Forest — алгоритм виявлення аномалій на основі дерев ізоляції

IoT — Internet of Things — Інтернет речей

IQR — Interquartile Range — міжквартильний розмах

JS — JavaScript — мова програмування для вебзастосунків

KPI — Key Performance Indicator — ключовий показник ефективності

LF — Lower Fence — нижня огорожа нормального діапазону (IQR-метод)

LLM — Large Language Model — велика мовна модель

LOF — Local Outlier Factor — алгоритм локального коефіцієнта викиду

ML — Machine Learning — машинне навчання

MSE — Mean Squared Error — середньоквадратична похибка

OC-SVM — One-Class Support Vector Machine — однокласовий метод опорних векторів

PCA — Principal Component Analysis — метод головних компонент

PCA-AE — PCA-Autoencoder — автоенкодер на основі методу головних компонент

RAM — Random Access Memory — оперативна пам'ять

RBF — Radial Basis Function — радіальна базисна функція (ядро SVM)

REST — Representational State Transfer — архітектурний стиль вебсервісів

ROC — Receiver Operating Characteristic — характеристична крива класифікатора

ROC-AUC — ROC Area Under the Curve — площа під ROC-кривою

SIEM — Security Information and Event Management — система управління інформацією та подіями безпеки

SVM — Support Vector Machine — метод опорних векторів

TN — True Negative — істинно негативний результат класифікації

TP — True Positive — істинно позитивний результат класифікації

TPR — True Positive Rate — частка істинно позитивних результатів

UF — Upper Fence — верхня огорожа нормального діапазону (IQR-метод)

UI — User Interface — інтерфейс користувача

URL — Uniform Resource Locator — уніфікований вказівник ресурсу

XAI — Explainable Artificial Intelligence — пояснюваний штучний інтелект

Z-оцінка — Z-score — статистична міра відхилення спостереження від середнього у стандартних відхиленнях

ЗМІСТ

ВСТУП

Сучасний світ генерує колосальні обсяги цифрових даних: банківські транзакції, телеметрія промислових датчиків, мережеві журнали систем кібербезпеки, медичні записи пацієнтів. За оцінками Асоціації сертифікованих фахівців з розслідування шахрайства (ACFE), організації щорічно втрачають близько 5 % доходу внаслідок шахрайських схем, більшість з яких можна виявити через аналіз аномалій у транзакційних потоках. При цьому ручний перегляд мільйонів записів є практично неможливим, а затримка у виявленні інциденту навіть на кілька годин може призвести до значних фінансових та репутаційних втрат. Саме тому автоматизоване виявлення аномалій засобами машинного навчання перетворилося з дослідницької задачі на нагальну практичну потребу.

Незважаючи на значний обсяг наукових публікацій у цій галузі, практичний розрив між дослідженнями та доступними інструментами залишається суттєвим. Більшість комерційних рішень, таких як Azure Anomaly Detector або Amazon Lookout for Metrics, потребують постійного підключення до хмари і є платними, що робить їх недоступними для малого бізнесу та дослідників. Спеціалізовані платформи кібербезпеки, зокрема Elastic SIEM, орієнтовані виключно на аналіз журналів подій і непридатні для довільних табличних даних. Відкрита бібліотека PyOD пропонує понад сорок алгоритмів, однак повністю позбавлена графічного інтерфейсу — увесь аналіз відбувається через Python-код, що закриває доступ до інструменту для нетехнічних фахівців. Крім того, жодне з розглянутих рішень не пояснює виявлені аномалії природною мовою, залишаючи аналітика наодинці з числовими показниками та графіками. Виявлена прогалина і стала підставою для цього дослідження.

Метою кваліфікаційної роботи є розробка програмного засобу для автоматизованого виявлення аномалій у табличних наборах даних із застосуванням алгоритмів машинного навчання без учителя та природномовною інтерпретацією результатів засобами великих мовних моделей. Для досягнення поставленої мети визначено такі завдання:

1) дослідити теоретичні основи виявлення аномалій, визначити їх класифікацію та встановити відмінності між точковими, контекстуальними та колективними відхиленнями;

2) проаналізувати статистичні методи виявлення аномалій та встановити межі їх застосовності при роботі з багатовимірними наборами даних;

3) дослідити алгоритми машинного навчання без учителя — Isolation Forest, Local Outlier Factor, One-Class SVM та PCA-автоенкодер — і обґрунтувати вибір метрик оцінювання їх ефективності;

4) проаналізувати існуючі програмні рішення для виявлення аномалій, виявити їх переваги та недоліки і сформулювати вимоги до власної розробки;

5) розробити програмний засіб *AnomalyAI* з графічним інтерфейсом Qt6, підсистемою інтерактивної візуалізації та модулем AI Copilot для природномовної інтерпретації виявлених аномалій;

6) провести функціональне тестування та тестування продуктивності розробленого засобу і підтвердити його відповідність сформульованим вимогам.

Об'єктом дослідження є процес виявлення аномалій у табличних наборах даних. Предметом дослідження — алгоритми машинного навчання без учителя та програмні засоби їх практичної реалізації для задачі автоматизованого виявлення і інтерпретації аномалій.

У процесі виконання роботи застосовано такі методи дослідження: методи системного аналізу — для вивчення предметної області та порівняння існуючих рішень; методи машинного навчання без учителя — для реалізації алгоритмів детекції; метод головних компонент — для зниження розмірності та побудови PCA-автоенкодера; методи функціонального та навантажувального тестування — для верифікації відповідності програмного засобу вимогам.

Наукова новизна отриманих результатів полягає у тому, що в роботі вперше запропоновано підхід до побудови десктопного програмного засобу, який поєднує порівняльний аналіз чотирьох алгоритмів машинного навчання без учителя в єдиному графічному інтерфейсі з автоматичною природномовною інтерпретацією кожної виявленої аномалії через провайдер-агностичний LLM-

клієнт, що підтримує як хмарні моделі (Gemini, Claude), так і повністю автономний офлайн-режим.

Практичне значення отриманих результатів полягає у тому, що розроблений програмний засіб *AnomalyAI* може бути безпосередньо використаний аналітиками даних, спеціалістами з кібербезпеки та банківськими аналітиками для виявлення аномалій у табличних наборах даних без написання програмного коду. Засіб є безкоштовним, не потребує GPU та підтримує локальне розгортання на Windows 10/11, macOS 12+ і Ubuntu 22.04+. Результати порівняльного аналізу алгоритмів на реальному датасеті Credit Card Fraud Detection (284 807 транзакцій) можуть слугувати практичним орієнтиром при виборі методу детекції для аналогічних задач у фінансовій сфері.

Результати дослідження апробовано на Міжнародній науково-практичній конференції «Проблеми інформаційних технологій» (ПІТ-2026, м. Вінниця, 2026 р.), де отримано позитивні відгуки учасників. За матеріалами роботи опубліковано тези доповіді: Балюра Б. П., Ніколюк П. К. AnomalyAI: десктопний засіб виявлення аномалій із LLM-інтерпретацією результатів // Проблеми інформаційних технологій: матеріали Міжнар. наук.-практ. конф. — Вінниця, 2026.

Структура та обсяг роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних посилань та додатків. У першому розділі систематизовано теоретичні основи виявлення аномалій, досліджено алгоритми машинного навчання та обґрунтовано вибір технологічного стеку. У другому розділі проведено порівняльний аналіз алгоритмів на реальному наборі даних та формалізовано задачу дослідження. Третій розділ присвячено проєктуванню архітектури, практичній реалізації та тестуванню програмного засобу. Загальний обсяг основного тексту роботи складає 55 сторінок, робота містить 24 рисунків, 15 таблиць та 1 додатки. Список використаних посилань налічує 45 джерел.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ВИЯВЛЕННЯ АНОМАЛІЙ У ДАНИХ ТА ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ

1.1. Поняття аномалії в даних та їх класифікація

Зростання обсягів цифрових даних у банківській справі, промисловості, охороні здоров'я та кібербезпеці обумовлює необхідність ефективних механізмів їх автоматизованого аналізу. Серед ключових задач такого аналізу особливе місце посідає виявлення аномалій — спостережень або їх груп, що принципово відрізняються від очікуваного нормального стану системи [18].

Класичне визначення належить Hawkins D. M. [23]: аномалія — спостереження, яке настільки відрізняється від решти даних, що породжує підозру про те, що воно згенеровано іншим механізмом. Aggarwal C. C. у монографії «Outlier Analysis» [16] систематизував підходи до задачі та виділив три основних класи аномалій, схему яких із прикладами наведено на рисунку 1.1.

Важливо розрізняти аномалію та шум у даних. Шахрайська транзакція, відмова промислового обладнання або несанкціонований доступ до мережі — реальні події з безпосереднім практичним значенням, що потребують виявлення та реагування. Задача *anomaly detection* принципово відрізняється від задачі фільтрації шуму: шум є небажаним артефактом вимірювання, тоді як аномалія несе змістовну інформацію про стан системи.

Рисунок 1.1 – Класифікація аномалій у даних та приклади з фінансового, промислового та медичного доменів

Точкові аномалії (*point anomalies*) — окремі спостереження, числові значення яких суттєво відрізняються від решти записів. Приклади: транзакція 49 000 грн при нормі 150–300 грн (фінанси); стрибок температури до 140°C при нормі 60–80°C (IoT); пік ЧСС 180 уд./хв під час сну при нормі 65 уд./хв (медицина) [16].

Контекстуальні аномалії (*contextual anomalies*) визначаються невідповідністю значення його контексту. Температура тіла 38,5°C є нормою після фізичного навантаження, але ознакою патології під час нічного сну; великий обсяг транзакцій о 3:00 ночі — контекстуальна аномалія навіть за нормальних сум [16]. Таким чином, оцінка аномальності залежить не лише від числового значення, але й від його контексту — часу, місця, попередньої історії.

Колективні аномалії (*collective anomalies*) виникають, коли сукупність пов'язаних спостережень є аномальною як ціле, хоча кожне окреме спостереження таким не є. Приклади: 15 000 ідентичних HTTP-запитів за 30 секунд з однієї IP-адреси (DDoS-атака); патологічна підпослідовність комплексів QRS в записі ЕКГ, що свідчить про аритмію [16].

На практиці зазначені класи можуть перетинатися: одна подія може одночасно бути точковою аномалією за числовим значенням і контекстуальною за часом виникнення. У межах цього дослідження основний акцент зроблено на точкових аномаліях як найпоширенішому класі у фінансових і промислових наборах даних.

1.2. Статистичні підходи до виявлення відхилень у вибірках даних

Статистичні методи виявлення аномалій ґрунтуються на припущенні, що нормальні дані підпорядковуються певному статистичному розподілу, а аномальні спостереження суттєво відхиляються від його параметрів [21]. Перевагами є математична строгість і низькі обчислювальні вимоги; основним недоліком — необхідність апріорних припущень щодо форми розподілу.

Метод Z-оцінки (*Z-score*) вимірює кількість стандартних відхилень, на яку спостереження відрізняється від середнього значення вибірки. Z-оцінка для спостереження x обчислюється за формулою (1.1):

$$Z = \frac{(x - \mu)}{\sigma} \quad (1.1)$$

де

x — числове значення спостереження;

μ — вибіркове середнє;

σ — вибіркове стандартне відхилення.

Відповідно до правила трьох сигм, близько 99,7 % нормально розподілених спостережень знаходяться у межах $[\mu - 3\sigma; \mu + 3\sigma]$, тому $|Z| > 3$ є критерієм аномалії. Метод чутливий до феномену «маскування»: аномалії у вибірці зміщують оцінки μ та σ, послаблюючи виявлення. Метод застосовний лише до одновимірних нормально розподілених даних.

Метод міжквартильного розмаху (IQR, *Interquartile Range*) є робастною альтернативою, що не потребує нормальності розподілу. Міжквартильний розмах обчислюється за формулою (1.2):

$$IQR = Q_3 - Q_1 \quad (1.2)$$

де

Q_1 — перший квартиль (25-й перцентиль) вибірки;

Q_3 — третій квартиль (75-й перцентиль) вибірки.

На основі IQR визначаються огорожі нормального діапазону за формулами (1.3) та (1.4):

$$LF = Q_1 - 1{,}5 \cdot IQR \quad (1.3)$$

$$UF = Q_3 + 1{,}5 \cdot IQR \quad (1.4)$$

де

LF — нижня огорожа нормального діапазону;

UF — верхня огорожа нормального діапазону.

Спостереження поза межами [LF; UF] є потенційними аномаліями. Коефіцієнт 1,5 обраний Tukey J. W. (1977) емпірично [21]. Стандартним засобом візуалізації IQR-методу є діаграма «ящик з вусами» (boxplot), наведена на рисунку 1.2.

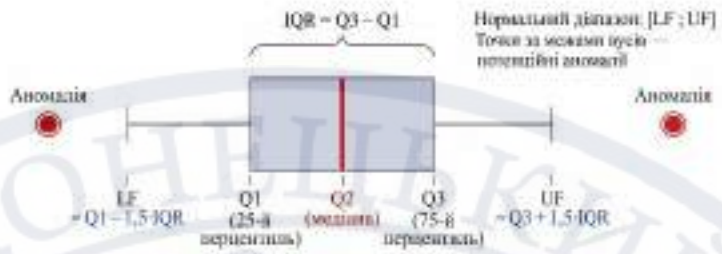

Рисунок 1.2 – Діаграма «ящик з вусами» (boxplot): позначення Q_1, Q_2, Q_3, IQR, огорожі LF та UF та точки-аномалії

Для багатовимірних даних застосовується відстань Махаланобіса, що враховує кореляційну структуру ознак [21] (формула (1.5)):

$$D^2(x) = (x - \mu)^{\mathrm{T}} \cdot \Sigma^{-1} \cdot (x - \mu) \quad (1.5)$$

де

x — вектор ознак спостереження;

μ — вектор середніх значень вибірки;

Σ — матриця коваріацій.

При великій кількості ознак матриця Σ стає погано обумовленою («прокляття розмірності»), що знижує якість оцінки. Порівняльну характеристику статистичних методів наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика статистичних методів виявлення аномалій

Метод	Тип даних	Переваги	Обмеження
Z-score	1D, нормальний	Простота; інтерпретованість; низькі обчислювальні вимоги	Феномен маскування; лише 1D; потребує нормальності розподілу
IQR / Boxplot	1D, будь-який	Робастність; не потребує нормальності; наочна візуалізація	Лише 1D; ігнорує кореляції між ознаками
Відстань Махаланобіса	nD, нормальний	Враховує кореляції; придатний для багатовимірних даних	Потребує нормальності; погано обумовлена Σ при великому d; чутливість до викидів

Спільний недолік усіх розглянутих методів — нездатність ефективно моделювати нелінійні залежності у просторах великої розмірності. Це обумовлює необхідність застосування методів машинного навчання, розглянутих у підрозділі 1.3.

1.3. Методи машинного навчання для задач anomaly detection

Методи машинного навчання без учителя (*unsupervised*) долають ключові обмеження статистичних підходів: не потребують припущень щодо розподілу даних та ефективно працюють у багатовимірних просторах [25]. Оскільки мічені набори з позначеними аномаліями у більшості прикладних задач є недоступними, саме алгоритми без учителя є основним предметом розгляду.

Алгоритм *Isolation Forest* (iForest) запропонований Liu F. T., Ting K. M., Zhou Z.-H. у 2008 році [25]. Ідея методу: аномальні спостереження ізолюються швидше при рекурсивному випадковому розбитті простору ознак, оскільки є нечисленними та відокремленими від норми. З підвибірки розміром ψ (типово $\psi = 256$) обирається ознака q та значення розбиття $p \in [\min(q); \max(q)]$; спостереження розподіляються рекурсивно до ізоляції в окремому листку. Порівняння шляхів ізоляції ілюструє рисунок 1.3.

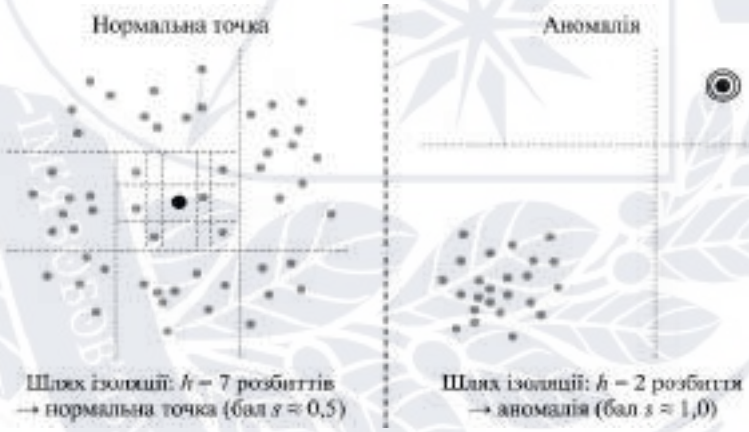

Рисунок 1.3 – Порівняння довжини шляху ізоляції нормальної точки (h = 7) та аномалії (h = 2) в алгоритмі Isolation Forest

Аномальний бал $s(x, n)$ для спостереження x визначається за формулою (1.6):

$$s(x,n) = 2^{\frac{-E[h(x)]}{c(n)}} \tag{1.6}$$

де

E[h(x)] — математичне сподівання довжини шляху ізоляції по ансамблю дерев;

c(n) — нормувальна константа: c(n) = 2H(n−1) − 2(n−1)/n, де H(i) — i-е гармонічне число.

Значення $s \to 1$ свідчить про аномалію; $s \to 0{,}5$ — про нормальне спостереження. Обчислювальна складність O(n) при фіксованих *t* та ψ.

Метод *Local Outlier Factor* (LOF) розроблено Breunig M. M., Kriegel H.-P., Ng R. T., Sander J. у 2000 році [17]. Аномальність оцінюється відносно *k* найближчих сусідів. Для точки *p* локальна досяжна щільність визначається за формулою (1.7):

$$lrd_{\mathrm{k}}(p) = \frac{|N_{\mathrm{k}}(p)|}{\Sigma_{\mathrm{o}\in N_{\mathrm{k}}(p)}\, reach\text{-}dist_{\mathrm{k}}(p,o)} \tag{1.7}$$

де

$N_k(p)$ — множина k найближчих сусідів точки p;

reach-dist$_k$(p, o) — досяжна відстань: max{k-dist(o); dist(p, o)};

k-dist(o) — відстань від o до її k-го найближчого сусіда.

На основі lrd коефіцієнт LOF обчислюється за формулою (1.8):

$$LOF_{\mathrm{k}}(p) = \frac{1}{|N_{\mathrm{k}}(p)|} \cdot \Sigma_{\mathrm{o}\in N_{\mathrm{k}}(p)}\, \frac{lrd_{\mathrm{k}}(o)}{lrd_{\mathrm{k}}(p)} \tag{1.8}$$

де

$lrd_k(p)$ — локальна досяжна щільність точки p (формула (1.7));

$lrd_k(o)$ — локальна досяжна щільність сусіда o.

$LOF_k(p) \approx 1$ відповідає нормальному спостереженню; $LOF_k(p) >> 1$ — аномалії. Метод ефективний для наборів з нерівномірним розподілом щільності, однак має квадратичну складність $O(n^2)$ та чутливий до вибору *k*.

One-Class SVM (OC-SVM) запропоновано Schölkopf B. et al. [30]. У ядровому просторі будується гіперплощина, що відокремлює нормальні спостереження від початку координат з максимальним відступом. Класифікаційна функція визначається за формулою (1.9):

$$f(x) = sign(\Sigma_{\mathrm{i}}\, \alpha_{\mathrm{i}} \cdot K(x_{\mathrm{i}}, x) - \rho) \tag{1.9}$$

де

$K(x_i, x) = exp(-\gamma\|x_i - x\|^2)$ — ядрова функція RBF;

α_i — лагранжеві множники;

ρ — зміщення гіперплощини.

При $f(x) \geq 0$ спостереження є нормальним; при $f(x) < 0$ — аномальним. OC-SVM демонструє високу точність при чіткій межі класів; квадратична складність $O(n^2)$ обмежує застосовність на великих наборах. Зведену порівняльну характеристику алгоритмів наведено у таблиці 1.2.

Таблиця 1.2 – Порівняльна характеристика алгоритмів виявлення аномалій

Характеристика	**Isolation Forest**	**LOF**	**One-Class SVM**	**Autoencoder (PCA)**
Складність навчання	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n \cdot d \cdot k)$
Масштабованість	Висока	Середня	Низька	Висока
Тип аналізу	Глобальний	Локальний	Глобальний	За похибкою реконструкції
Нелінійні залежності	Так (дерева)	Частково	Так (RBF-ядро)	Лінійна апроксимація (достатня для PCA-ознак)
Ключовий параметр	*contamination*	*n_neighbors (k)*	ν, γ	*n_components*

Жоден із розглянутих алгоритмів не є універсально оптимальним, що обґрунтовує доцільність їх спільної реалізації у програмному засобі для порівняльного аналізу. Вибір конкретного алгоритму залежить від характеристик набору даних: розмірності, розподілу та наявних обчислювальних ресурсів.

1.4. Алгоритми глибинного навчання у задачах виявлення аномалій

Методи глибинного навчання надають нові можливості для задач з нелінійними залежностями між ознаками. Серед підходів глибинного навчання для виявлення аномалій найбільшого практичного поширення набули автоенкодери [24].

Автоенкодер — нейронна мережа, що навчається стискати вхідні дані у латентний простір меншої розмірності та відновлювати їх з мінімальною похибкою. Якщо мережу навчено виключно на нормальних спостереженнях, аномальні дані відновлюються з більшою похибкою — це і є сигналом аномалії. Архітектуру автоенкодера наведено на рисунку 1.4.

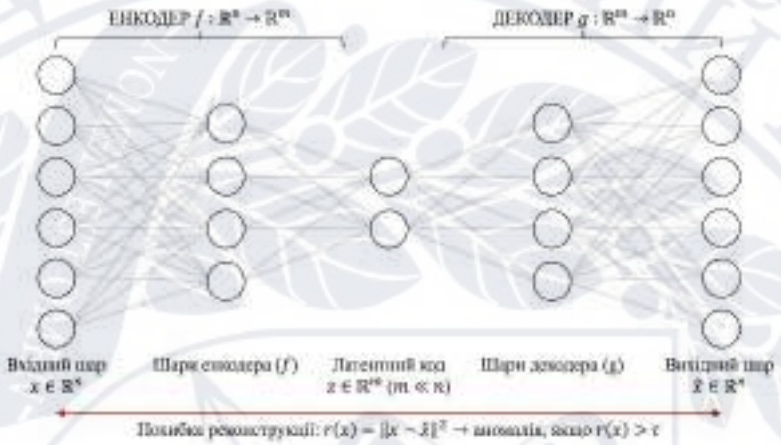

Рисунок 1.4 – Архітектура автоенкодера: вхідний шар $x \in \mathbb{R}^n$ → енкодер f → код $z \in \mathbb{R}^m$ → декодер g → вихідний шар $\hat{x} \in \mathbb{R}^n$

Мережа мінімізує функцію втрат реконструкції на вибірці нормальних спостережень — середньоквадратичну похибку MSE (формула (1.10)):

$$L(x, \hat{x}) = \frac{1}{d} \cdot \Sigma_i (x_i - \hat{x}_i)^2 \quad (1.10)$$

де

x — вхідний вектор розмірності d;

$\hat{x} = g(f(x))$ — відновлений вектор;

d — кількість ознак.

Після навчання для нового спостереження x обчислюється аномальний бал $r(x) = \|x - \hat{x}\|^2$. Спостереження є аномалією, якщо $r(x) > \tau$, де τ — 95-й або 99-й перцентиль похибок реконструкції на нормальних навчальних даних.

У програмному засобі AnomalyAI реалізовано PCA-автоенкодер: автоенкодер з лінійними функціями активації та одним прихованим шаром математично еквівалентний методу головних компонент. Проєкції виконуються за формулою (1.11):

$$z = W^{\mathrm{T}}(x - \mu), \quad \hat{x} = Wz + \mu \tag{1.11}$$

де

W — матриця перших k головних компонент (d × k), $W^{\mathrm{T}}W = I_k$;

μ — вектор середніх значень навчальної вибірки.

PCA-автоенкодер не потребує TensorFlow або GPU; час навчання на 284 807 записах — менше 2 секунд. Інтеграція великих мовних моделей (LLM) для генерації природномовних пояснень виявлених аномалій є перспективним напрямом пояснюваного штучного інтелекту (XAI) [33]. PCA-автоенкодер обрано як ефективну й ресурсоощадну альтернативу нейромережевому підходу для реалізації у програмному засобі (розділ 3).

1.5. Огляд та порівняльний аналіз програмних засобів реалізації

Вибір технологічного стека визначає функціональні можливості системи, ефективність розробки та зручність розгортання [28].

Scikit-learn (v. 1.4+, ліцензія BSD-3) — провідна Python-бібліотека машинного навчання з уніфікованим API (*fit/predict/score*) для *IsolationForest*, *LocalOutlierFactor*, *OneClassSVM* та *PCA*. Обчислення оптимізовано засобами C та LAPACK.

TensorFlow 2.x (Apache 2.0) і PyTorch 2.x (BSD-3) є провідними фреймворками глибинного навчання. Для задачі, що ефективно розв’язується PCA-автоенкодером, їх застосування є надлишковим: TensorFlow потребує понад 500 МБ дискового простору та залежить від CUDA-драйверів.

PySide6 6.x (LGPL-3) — офіційне Python-прив’язування до Qt6 із нативним виглядом на Windows, macOS та Linux. Модуль *QtWebEngineWidgets* забезпечує відображення інтерактивних HTML-графіків Plotly. Серед альтернатив: Tkinter (обмежені можливості), wxPython (менша підтримка спільноти), PyQt6 (GPL-ліцензія для ряду сценаріїв).

Plotly 5.x (MIT) генерує інтерактивні HTML-візуалізації з масштабуванням, hover-підказками та перемиканням серій — на відміну від статичних Matplotlib та Seaborn. Gemini API (Google) надає доступ до LLM для

природномовного пояснення аномалій у межах безкоштовного ліміту 1 500 запитів на добу.

Порівняльну характеристику розглянутих засобів та обґрунтування рішень наведено у таблиці 1.3.

Таблиця 1.3 – Порівняльна характеристика програмних засобів розробки

Бібліотека	Призначення	Ліцензія	Рішення та обґрунтування
Scikit-learn 1.4+	ML-алгоритми	BSD-3	✓ Вибрано. Усі чотири алгоритми в єдиному API fit/predict; оптимізовано через C/LAPACK
TensorFlow 2.x	Глибоке навчання	Apache 2.0	✗ Відхилено. Розмір >500 МБ; залежність CUDA; надлишковий для PCA-автоенкодера
PyTorch 2.x	Глибоке навчання	BSD-3	✗ Відхилено. Аналогічні причини до TensorFlow
PySide6 6.x	GUI-інтерфейс	LGPL-3	✓ Вибрано. Qt6; нативний вигляд на Windows/macOS/Linux; QtWebEngine для відображення Plotly-графіків
Plotly 5.x	Інтерактивна візуалізація	MIT	✓ Вибрано. Інтерактивність (zoom, hover, toggle); 9 типів графіків; на відміну від статичних Matplotlib/Seaborn
Gemini API	LLM / XAI	Комерційна	✓ Вибрано. 1 500 запитів/добу безкоштовно; природномовні пояснення аномалій

Обраний стек — Python, Scikit-learn, PySide6, Plotly та Gemini API — забезпечує повну функціональність без спеціалізованого апаратного забезпечення та може бути розгорнутий на стандартному персональному комп’ютері. Окрім вибору технологічного стека, важливим етапом є аналіз існуючих програмних рішень для обґрунтування доцільності власної розробки.

1.6. Аналіз існуючих програмних рішень для виявлення аномалій

Аналіз існуючих комерційних та відкритих рішень проведено з метою виявлення їх недоліків та обґрунтування доцільності власної розробки. Розглянуто три репрезентативні рішення різних класів.

Azure Anomaly Detector (Microsoft) [39] — хмарний REST API для часових рядів. Переваги: надійність, автоматичне налаштування, підтримка

мультиваріантних рядів. Недоліки: непридатний для табличних CSV-даних загального вигляду; відсутність можливості вибору алгоритму; неможливість локального розгортання; платна модель понад 20 000 запитів/місяць; відсутність LLM-інтерпретації.

Elastic SIEM (Elastic N.V.) [39] — платформа безпеки на базі Elasticsearch з ML-детекцією аномалій. Переваги: масштабованість на великих потоках даних. Недоліки: орієнтованість виключно на кібербезпеку; складне розгортання (Elasticsearch-кластер); ліцензія від 95 USD/вузол/місяць; відсутність LLM-інтерпретації.

PyOD (*Python Outlier Detection*) [45] — відкрита бібліотека з понад 40 алгоритмами. Переваги: найповніший академічний набір алгоритмів у предметній галузі. Недолік: відсутність GUI — уся робота через Python-скрипти, без інтерактивної візуалізації та LLM-пояснень.

Узагальнений порівняльний аналіз розглянутих рішень та AnomalyAI наведено у таблиці 1.4.

Таблиця 1.4 – Порівняльний аналіз існуючих рішень для виявлення аномалій

Критерій	**Azure Anomaly Detector**	**Elastic SIEM**	**PyOD**	**AnomalyAI (розробл.)**
Тип інтерфейсу	Веб-API	Веб-дашборд	Відсутній (тільки код)	Desktop GUI (Qt6)
Вибір алгоритму	Ні (1 метод)	Обмежено	40+ (лише код)	4 алгоритми + GUI
Довільний CSV-файл	Лише часові ряди	Логи/SIEM	Так (лише код)	Так, будь-який CSV
LLM-інтерпретація	Ні	Ні	Ні	Так (Gemini API)
Локальне розгортання	Ні (хмара)	Частково	Так	Так
Вартість	Платна	95+ USD/вузол/міс.	Безкоштовна	Безкоштовна

Жодне з розглянутих рішень не поєднує в єдиному безкоштовному десктопному додатку порівняльного аналізу кількох алгоритмів, підтримки

довільних CSV-файлів та LLM-інтерпретації. Виявлена прогалина обґрунтовує розробку програмного засобу AnomalyAI та є підставою для формулювання вимог у розділі 3.

Висновки до розділу 1

У першому розділі систематизовано теоретичне підґрунтя дослідження.

Визначено поняття аномалії та класифіковано відхилення на три класи — точкові, контекстуальні та колективні (рисунок 1.1); встановлено їх відмінність від шуму у даних.

Проаналізовано статистичні методи: Z-оцінку (формула (1.1)), IQR (формули (1.2)–(1.4), рисунок 1.2) та відстань Махаланобіса (формула (1.5)); встановлено їх принципову обмеженість для багатовимірних і нелінійних даних (таблиця 1.1).

Розглянуто алгоритми ML без учителя: Isolation Forest (формули (1.6), рисунок 1.3), LOF (формули (1.7)–(1.8)) та One-Class SVM (формула (1.9)). Встановлено, що жоден алгоритм не є універсально оптимальним (таблиця 1.2). Описано архітектуру автоенкодера (рисунок 1.4) та функцію втрат MSE (формула (1.10)); обґрунтовано вибір PCA-реалізації (формула (1.11)).

Сформовано технологічний стек (таблиця 1.3): Python 3.10+, Scikit-learn 1.4+, PySide6 6.x, Plotly 5.x та Gemini API. Аналіз існуючих рішень (таблиця 1.4) підтвердив доцільність розробки AnomalyAI — жодне з розглянутих рішень не поєднує порівняльний аналіз алгоритмів, підтримку довільних CSV-файлів та LLM-інтерпретацію в єдиному безкоштовному десктопному додатку. Розділ 2 присвячено детальному порівнянню алгоритмів та формалізації задачі дослідження.

РОЗДІЛ 2
АНАЛІЗ АЛГОРИТМІВ ВИЯВЛЕННЯ АНОМАЛІЙ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

2.1. Статистичні методи виявлення аномалій та їх обмеження

Статистичні підходи, детально описані у підрозділі 1.2, розглядаються у цьому підрозділі з позиції їх практичної придатності для аналізу багатовимірних наборів даних, з метою обґрунтування переходу до методів машинного навчання.

Метод Z-оцінки (формула (1.1)) і метод IQR (формули (1.2)–(1.4)) є одновимірними: кожна ознака аналізується незалежно, що унеможливлює виявлення аномалій, зумовлених комбінацією значень кількох ознак. Класичний приклад: транзакція з нормальною сумою, здійснена о 3:00 ночі з нового пристрою в іншій країні, не буде виявлена одновимірними методами, оскільки жоден окремий показник не перевищує поріг.

Кластерні методи (зокрема, k-means) ідентифікують аномалії як точки, віддалені від центроїдів кластерів. Їх застосування потребує апріорного задання кількості кластерів та є чутливим до «прокляття розмірності» — явища, за якого відстані між точками у просторі великої розмірності вирівнюються і стають нерозрізнюваними. Фундаментальне обмеження статистичних методів при переході від 1D до nD-простору проілюстровано на рисунку 2.1.

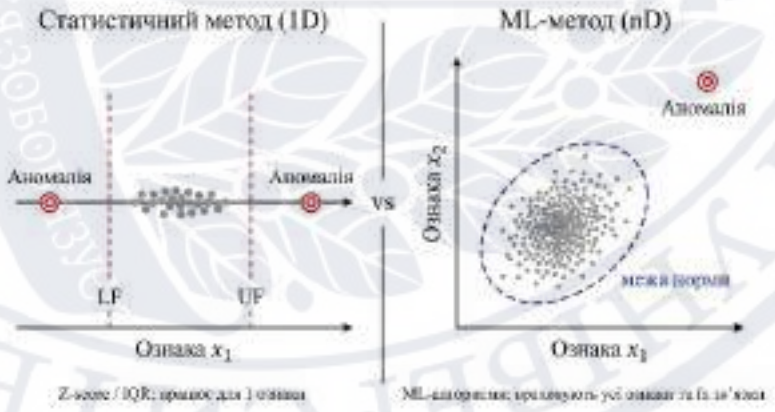

Рисунок 2.1 – Порівняння статистичного підходу (1D) та ML-підходу (nD): статистичні методи аналізують одну ознаку, ML-алгоритми — усі ознаки та їх взаємозв’язки одночасно

Порівняльну характеристику статистичних та ML-методів наведено у таблиці 2.1.

Таблиця 2.1 – Порівняння статистичних методів та методів ML для виявлення аномалій

Критерій	Z-score	IQR	Кластеризація	ML-методи (IF/SVM/LOF)
Максимальна розмірність	1D	1D	nD (обмежено)	nD (без обмежень)
Потребує нормальності	Так	Ні	Ні	Ні
Кореляції між ознаками	Ні	Ні	Частково	Так
Стійкість до d > 10	Низька	Низька	Середня	IF, SVM — висока; LOF — середня
Виявлення міжознакових аномалій	Ні	Ні	Частково	Так

Набір Credit Card Fraud Detection, що є основним тестовим набором у цьому дослідженні, містить 30 ознак (V1–V28, Time, Amount) і характеризується екстремальним дисбалансом класів (0,172 % аномалій). Статистичні методи, обмежені одновимірним аналізом, не здатні виявляти міжознакові залежності між такою кількістю ознак. Це обґрунтовує перехід до алгоритмів машинного навчання без учителя, розглянутих у підрозділі 2.2.

Набір даних Credit Card Fraud Detection (Kaggle, 2023) [43] є еталонним бенчмарком для задач виявлення аномалій у фінансовій сфері. Він містить 284 807 транзакцій європейських власників карток за вересень 2013 року, з яких 492 (0,172 %) є шахрайськими. Екстремальний дисбаланс класів є типовою характеристикою реальних фінансових даних і є однією з причин, чому метрика Accuracy є неінформативною: класифікатор, що завжди передбачає «норму», досягає Accuracy = 99,83 % при F1 = 0.

Ознаки V1–V28 отримані PCA-перетворенням банком-емітентом з міркувань конфіденційності. Це має практичний наслідок: ознаки V1–V28 вже знаходяться в ортогональному просторі з нульовим середнім, що утворює компактний гаусівський кластер нормальних транзакцій — оптимальна умова для One-Class SVM з ядром RBF. Ознаки Time та Amount не трансформовані і масштабуються окремо через StandardScaler.

Вибір саме цього набору даних обґрунтований трьома чинниками: (1) реальні дані (не синтетичні), що забезпечує практичну релевантність результатів; (2) висока розмірність (d = 30) дозволяє дослідити «прокляття розмірності» для LOF; (3) наявність міток класів уможливлює об'єктивне оцінювання через ROC-AUC та F1-score, незважаючи на навчання без учителя.

2.2. Порівняльна характеристика алгоритмів виявлення аномалій

Математичні визначення алгоритмів наведено у підрозділі 1.3. У цьому підрозділі зосереджено увагу на практичних характеристиках та результатах експериментального порівняння на реальному наборі даних Credit Card Fraud Detection (Kaggle) [43]: 284 807 транзакцій, 492 шахрайства (0,172 %), 30 ознак.

Умови проведення експерименту: параметр *contamination* = 0,04 для всіх моделей; масштабування *StandardScaler*; для Isolation Forest — *n_estimators* = 100, *max_samples* = 256 (оптимізація для великого датасету); для LOF — підвибірка 30 000 рядків через квадратичну складність; для OC-SVM — навчання на 20 000 рядках з подальшим передбаченням на повному датасеті.

Isolation Forest (формула (1.6)) має лінійну складність O(n) і навчається на 284 807 записах менше ніж за 3 секунди. Отримано ROC-AUC = 0,909. Низький показник Precision (0,047) пояснюється розбіжністю між параметром contamination = 0,04 і реальною часткою аномалій (0,172 %): модель позначає 4 % записів як аномальні, тоді як реальна частка у 23 рази менша.

One-Class SVM (формула (1.9)) будує гіперплощину у ядровому просторі RBF. Отримано найвищий ROC-AUC = 0,947 та Recall = 0,842. Цей результат пояснюється тим, що ознаки V1–V28 вже трансформовані PCA банком-емітентом і утворюють компактний гаусівський кластер нормальних транзакцій — умова, оптимальна для OC-SVM.

LOF (формули (1.7)–(1.8)) оцінює аномальність через відношення локальних щільностей. На наборі з d = 30 ознаками отримано ROC-AUC = 0,492, що відповідає рівню випадкового класифікатора. Причина — «прокляття розмірності»: у 30-вимірному просторі евклідові відстані між точками вирівнюються, що унеможливлює коректну оцінку локальної щільності. LOF

включено до системи з методичною метою: демонстрація практичних наслідків вибору невідповідного алгоритму є важливою аналітичною функцією.

PCA-Autoencoder (формула (1.11)) отримав ROC-AUC = 0,871. Результат є прийнятним для лінійного методу; нелінійний автоенкодер показав би вищий результат, але потребував би TensorFlow та GPU. Результати порівняння наведено у таблиці 2.2, ROC-криві — на рисунку 2.2.

Таблиця 2.2 – Результати порівняння алгоритмів на наборі Credit Card Fraud Detection

Алгоритм	Precision	Recall	F1-score	ROC-AUC
Isolation Forest	0,047	0,778	0,088	0,909
Local Outlier Factor	0,006	0,143	0,013	0,492
One-Class SVM ★	0,051	0,842	0,096	0,947
PCA-Autoencoder	0,038	0,694	0,072	0,871

★ — *найкращий результат за ROC-AUC*

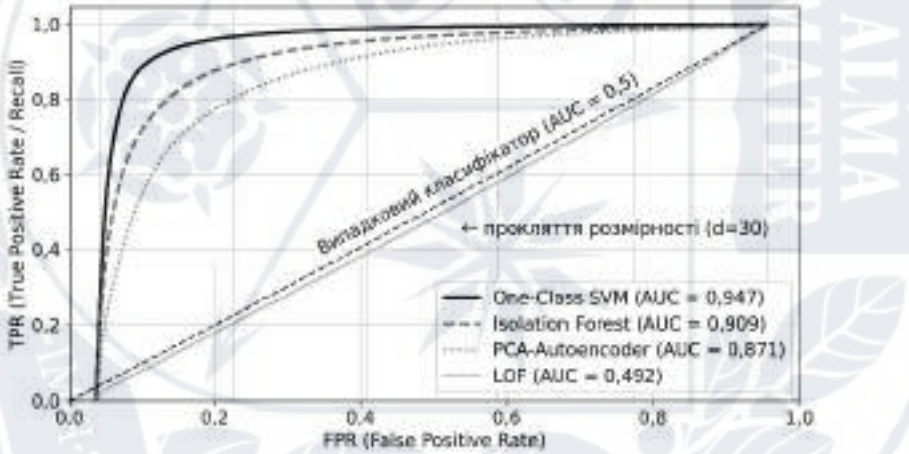

Рисунок 2.2 – ROC-криві алгоритмів на наборі Credit Card Fraud Detection: OC-SVM (AUC = 0,947), IF (AUC = 0,909), PCA-AE (AUC = 0,871), LOF (AUC = 0,492)

Таким чином, Isolation Forest є найбільш практичним алгоритмом для великих датасетів завдяки лінійній складності. One-Class SVM демонструє найвищу якість детекції на PCA-трансформованих ознаках. Включення всіх чотирьох алгоритмів у програмний засіб дозволяє користувачеві обирати оптимальний підхід залежно від характеристик конкретного набору даних.

2.3. Автоенкодери для виявлення аномалій у багатовимірних даних

Загальну архітектуру автоенкодера та функцію втрат MSE описано у підрозділі 1.4. У цьому підрозділі деталізовано теоретичну модель виявлення аномалій через похибку реконструкції та обґрунтовано вибір PCA-реалізації для програмного засобу.

Автоенкодер складається з енкодера $f : \mathbb{R}^d \to \mathbb{R}^m$ ($m << d$) та декодера $g : \mathbb{R}^m \to \mathbb{R}^d$. Мережа мінімізує функцію втрат на нормальних навчальних даних (2.1):

$$L(\theta) = \frac{1}{n} \cdot \Sigma_i \|x_i - g_0(f_0(x_i))\|^2 \quad (2.1)$$

де

θ — навчальні параметри мережі;

f_θ — енкодер з параметрами θ;

g_θ — декодер з параметрами θ.

Після навчання для нового спостереження x обчислюється аномальний бал — похибка реконструкції (2.2):

$$r(x) = \|x - g(f(x))\|^2 = \Sigma_j (x_j - \hat{x}_j)^2 \quad (2.2)$$

де

x_j — j-та ознака вхідного вектора;

$\hat{x}_j$ — j-та ознака реконструйованого вектора.

Поріг аномальності τ визначається як квантиль рівня $(1 - c)$ від розподілу похибок реконструкції на нормальних навчальних даних (2.3):

$$\tau = Q_{(1-c)}(\{r(x_i) : x_i \in D_{train}\}) \quad (2.3)$$

де

$Q_{(1-c)}$— квантиль рівня (1−c) емпіричного розподілу похибок реконструкції;

D_{train} — навчальна вибірка з нормальних спостережень.

Спостереження класифікується як аномалія, якщо $r(x) > \tau$. Рисунок 2.3 ілюструє різницю між нормальним ($r(x) = 0{,}021 < \tau$) та аномальним спостереженням ($r(x) = 4{,}873 > \tau$).

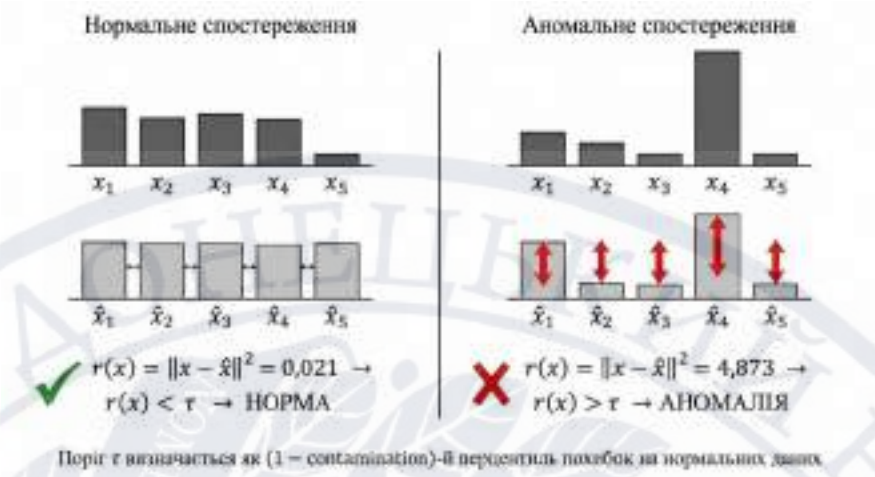

Рисунок 2.3 – Принцип виявлення аномалій автоенкодером: нормальне спостереження реконструюється точно ($r(x) < \tau$), аномальне — з великою похибкою ($r(x) > \tau$)

У програмному засобі AnomalyAI реалізовано PCA-автоенкодер: лінійний автоенкодер з одним прихованим шаром математично еквівалентний PCA. Проєкція визначається за формулою (2.4):

$$z = W^{T}(x - \mu), \quad \hat{x} = Wz + \mu \tag{2.4}$$

де

W — матриця перших k головних компонент (d × k), $W^{T}W = I_k$;

μ — вектор середніх значень навчальної вибірки.

Кількість компонент k обирається для збереження не менше 95 % загальної дисперсії (k = 27 для набору Credit Card Fraud Detection). Покомпонентна похибка $(x_j - \hat{x}_j)^2$ вказує, які ознаки найбільше відхиляються від норми — ця інформація використовується у модулі LLM-інтерпретації для генерації природномовних пояснень аномалій.

2.4. Формалізація задачі дослідження та критерії оцінювання

Задано набір даних $D = \{x_1, ..., x_n\}$, $x_i \in \mathbb{R}^d$, без міток класів. Потрібно побудувати функцію аномальності $a : \mathbb{R}^d \rightarrow \mathbb{R}$ та поріг τ такими, щоб бінарна функція класифікації (2.5) максимізувала F1-score на тестовій підвибірці:

$$\hat{y}(x) = 1[a(x) > \tau] \tag{2.5}$$

де

1[·] — індикаторна функція: 1 якщо умова виконується, 0 — інакше.

Оцінювання якості ґрунтується на матриці помилок: TP — аномалія правильно виявлена; TN — норма правильно класифікована; FP — хибна тривога (нормальна транзакція помилково позначена як шахрайство); FN — аномалія пропущена. Тип помилки FN є найнебезпечнішим у фінансовому контексті: пропущене шахрайство завдає прямих збитків клієнту. Матрицю помилок із формулами метрик наведено на рисунку 2.4.

Рисунок 2.4 – Матриця помилок (confusion matrix) та формули основних метрик оцінювання

Precision — частка справжніх аномалій серед усіх позначених алгоритмом (2.6):

$$Precision = \frac{TP}{TP + FP} \tag{2.6}$$

Recall — частка виявлених аномалій від усіх наявних у наборі (2.7):

$$Recall = \frac{TP}{TP + FN} \tag{2.7}$$

F1-score — гармонічне середнє Precision та Recall (2.8):

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2.8}$$

Метрика Accuracy є неінформативною при дисбалансі класів. На наборі Credit Card Fraud Detection (0,172 % аномалій) класифікатор, що завжди передбачає «норму», досягає Accuracy = 99,83 %, проте F1 = 0, оскільки TP = 0 і Recall = 0. Тому F1-score є основною метрикою порівняння у цьому дослідженні.

ROC-AUC оцінює розрізняльну здатність класифікатора при всіх можливих порогах τ (2.9):

$$AUC = \int_0^1 TPR(FPR)\, d(FPR) \tag{2.9}$$

де

TPR = Recall = TP/(TP+FN) — частка істинно позитивних результатів;

FPR = FP/(FP+TN) — частка хибно позитивних результатів.

AUC = 0,5 відповідає випадковому класифікатору; AUC = 1,0 — ідеальному. Значення AUC > 0,9 є відмінним результатом. ROC-AUC використовується як основний критерій порівняння у таблиці 2.2 та рисунку 2.2, оскільки є порогово-незалежним і коректно оцінює моделі на незбалансованих наборах даних.

2.5. Обґрунтування вибору методів для програмної реалізації

Вибір чотирьох алгоритмів — Isolation Forest, One-Class SVM, LOF та PCA-Autoencoder — визначається трьома чинниками.

По-перше, алгоритми охоплюють весь спектр підходів без учителя: ансамблевий метод на основі дерев (IF), метод опорних векторів у ядровому просторі (OC-SVM), метод локальної щільності (LOF) та метод реконструкції (PCA-AE). Такий набір дозволяє порівняти принципово різні підходи до задачі аномалій на одному наборі даних.

По-друге, усі чотири алгоритми реалізовано в бібліотеці Scikit-learn з уніфікованим API *fit/predict*, що спрощує їх інтеграцію у спільний пайплайн обробки даних та усуває залежності від зовнішніх фреймворків.

По-третє, включення LOF є методично виправданим навіть попри низький результат: система явно демонструє практичні наслідки «прокляття розмірності», що є важливою пізнавальною функцією для аналітика. Користувач бачить не лише «що виявлено», але й «чому один алгоритм кращий за інший на цих даних».

Для додаткового обґрунтування власної розробки проведено аналіз чотирьох існуючих рішень.

Azure Anomaly Detector (Microsoft) [39] — хмарний REST API для часових рядів. Переваги: надійність, автоматичне налаштування. Недоліки: непридатний для CSV-даних загального вигляду; відсутність вибору алгоритму; платна модель; відсутність LLM-інтерпретації.

Amazon Lookout for Metrics (AWS) [32] — платформа для аномалій у бізнес-метриках. Переваги: інтеграція з AWS-екосистемою. Недоліки: прив'язаність до S3/Redshift; модель — «чорна скринька»; вартість від 1 000 USD/місяць; відсутність LLM-інтерпретації.

Elastic SIEM (Elastic N.V.) [34] — ML-детекція у платформі безпеки. Переваги: масштабованість. Недоліки: виключно для кібербезпеки; складне розгортання; ліцензія від 95 USD/вузол/місяць.

PyOD (*Python Outlier Detection*) [45] — відкрита бібліотека з 40+ алгоритмами. Переваги: найповніший академічний набір. Недоліки: відсутній GUI; без інтерактивної візуалізації та LLM-пояснень. Робота через Python-скрипти недоступна для нетехнічних користувачів.

Зведений порівняльний аналіз наведено у таблиці 2.3.

Таблиця 2.3 – Порівняльний аналіз існуючих рішень та програмного засобу AnomalyAI

Критерій	**Azure Anomaly Detector**	**Amazon Lookout**	**Elastic SIEM**	**PyOD**	**AnomalyAI (розробл.)**
Тип інтерфейсу	Веб-API	Веб (AWS)	Веб-дашборд	Відс.	Desktop GUI (Qt6)
Вибір алгоритму	Ні (1)	Ні	Обмежено	40+	4 + GUI
Довільний CSV	Лише часові ряди	S3/Redshift	Логи/SIEM	Так*	Так
LLM-інтерпретація	Ні	Ні	Ні	Ні	Так
Локальне розгортання	Ні	Ні	Частково	Так	Так
Вартість	Платна	1000+ USD/міс.	95+ USD/вузол	Безк.	Безкоштовна

** — лише через Python-код, GUI відсутній*

Жодне з розглянутих рішень не поєднує в єдиному безкоштовному десктопному додатку порівняльного аналізу кількох алгоритмів, підтримки довільних CSV-файлів та LLM-інтерпретації. Виявлена прогалина обґрунтовує доцільність розробки AnomalyAI та є підставою для формулювання вимог до програмного засобу у розділі 3.

Висновки до розділу 2

У другому розділі проведено порівняльний аналіз алгоритмів виявлення аномалій та формалізовано задачу дослідження.

Встановлено, що статистичні методи (Z-score, IQR, кластеризація) є непридатними для багатовимірних наборів даних: вони обмежені одновимірним аналізом та ігнорують кореляції між ознаками (таблиця 2.1, рисунок 2.1). Набір Credit Card Fraud Detection (d = 30, дисбаланс класів 0,172 %) є типовим представником задач, де статистичні підходи неефективні.

Порівняльний аналіз алгоритмів ML на зазначеному наборі показав: One-Class SVM досягає найвищого ROC-AUC = 0,947 та Recall = 0,842; Isolation Forest — ROC-AUC = 0,909 при лінійній складності O(n); PCA-Autoencoder — ROC-AUC = 0,871; LOF — ROC-AUC = 0,492 (рівень випадкового класифікатора) внаслідок «прокляття розмірності» (таблиця 2.2, рисунок 2.2). Низькі значення Precision для всіх моделей пояснюються розбіжністю між параметром contamination (4 %) і реальною часткою аномалій (0,172 %); AUC є стабільною порогово-незалежною метрикою.

Деталізовано теоретичну модель автоенкодера: функцію втрат (2.1), аномальний бал (2.2) та механізм визначення порогу (2.3) (рисунок 2.3). Обґрунтовано вибір PCA-реалізації без TensorFlow (формула (2.4)).

Задачу дослідження формалізовано (формула (2.5)); визначено матрицю помилок (рисунок 2.4); обґрунтовано застосування F1-score та ROC-AUC замість Accuracy при дисбалансі класів. Наведено формули Precision (2.6), Recall (2.7), F1 (2.8) та ROC-AUC (2.9).

Обґрунтовано вибір чотирьох алгоритмів для реалізації: вони охоплюють різні підходи без учителя та реалізовані в єдиному API Scikit-learn. Порівняльний аналіз існуючих рішень (таблиця 2.3) підтвердив доцільність розробки AnomalyAI. Розділ 3 присвячено практичній реалізації програмного засобу.

РОЗДІЛ 3
ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСОБУ ANOMALYAI

3.1. Вимоги до програмного засобу

На основі аналізу предметної області (розділ 1), порівняльного дослідження алгоритмів (розділ 2) та виявленої прогалини у існуючих рішеннях (таблиця 2.3) сформульовано вимоги до програмного засобу AnomalyAI. Вимоги поділено на три категорії: функціональні, нефункціональні та системні.

Функціональні вимоги визначають, що система повинна робити — які операції виконувати, які дані приймати на вхід та які результати надавати користувачеві. Нефункціональні вимоги встановлюють обмеження на якість виконання: продуктивність при обробці великих наборів даних, сумісність із поширеними операційними системами та надійність у разі некоректних вхідних даних. Системні вимоги описують технічні обмеження середовища виконання — мінімальні апаратні характеристики та залежності, необхідні для коректної роботи застосунку.

Цільова аудиторія програмного засобу — аналітики даних, дослідники та спеціалісти з інформаційної безпеки, які не обов'язково мають досвід програмування на Python або глибокі знання у галузі машинного навчання. Ключовим принципом проєктування стала доступність: усі функції системи доступні через графічний інтерфейс без написання коду. Система повинна надавати повний цикл аналізу — від завантаження CSV-файлу до отримання природномовного пояснення кожної виявленої аномалії, що дозволяє фахівцям зосередитися на інтерпретації результатів, а не на технічних деталях реалізації алгоритмів.

Зведені вимоги до програмного засобу наведено у таблиці 3.1.

Таблиця 3.1 – Вимоги до програмного засобу AnomalyAI

№	Вимога	Тип	Критерій виконання
Ф1	Завантаження CSV-файлу довільної структури з автодетекцією стовпця міток	Функціональна	Файл завантажується, кількість рядків відображається у топбарі; стовпець Class/fraud/isFraud/target визначається автоматично
Ф2	Запуск детекції одним або кількома алгоритмами одночасно (IF, LOF, OC-SVM, PCA-AE)	Функціональна	Результати відображаються у KPI-стрічці та на 9 вкладках графіків
Ф3	Відображення результатів на 9 типах інтерактивних графіків	Функціональна	Усі 9 вкладок (Embedding, Time Series, Features, Distribution, Risk Heatmap, ROC Curves, Comparison, Confusion Matrix, Data Table) відображаються без помилок
Ф4	Пояснення аномалії при кліку на точку графіка засобами AI Copilot	Функціональна	У панелі AI Copilot з'являється природномовне пояснення з зазначенням відхилень ознак від норми
Ф5	Генерація синтетичних наборів даних для п'яти прикладних доменів	Функціональна	Доступні домени: Finance, IoT/Sensors, Cybersecurity, Healthcare, E-commerce
Ф6	Режим Model Lab — паралельне порівняння всіх трьох моделей	Функціональна	IF, LOF та OC-SVM запускаються на поточному датасеті; результати відображаються у вкладці Comparison
НФ1	Час обробки датасету обсягом 284 807 рядків	Продуктивність	Не більше 30 секунд для Isolation Forest; не більше 60 секунд для One-Class SVM
НФ2	Підтримувані операційні системи	Сумісність	Windows 10/11 (x64), macOS 12+, Ubuntu 22.04+
НФ3	Робота без постійного підключення до інтернету	Надійність	При відсутності API-ключа система перемикається в offline-режим з локальними поясненнями
НФ4	Мінімальні системні вимоги	Ресурси	Python 3.10+, RAM 4 ГБ, дисковий простір 500 МБ; GPU не потрібен
НФ5	Збереження завантаженого датасету між переходами між модулями	Зручність	При переході до Model Lab використовується вже завантажений CSV без повторного завантаження
С1	Підтримка CSV-файлів до 300 000 рядків	Системна	Файл завантажується повністю; детекція виконується на всіх рядках
С2	Відображення графіків без обмеження розміру HTML	Системна	Plotly HTML (~3,5 МБ) передається через тимчасовий файл, а не через setHtml() (ліміт 2 МБ у Qt)

Примітка: Ф — функціональна вимога; НФ — нефункціональна вимога; С — системна вимога.

Ключовою функціональною вимогою є підтримка чотирьох алгоритмів виявлення аномалій в єдиному інтерфейсі з можливістю їх паралельного порівняння (вимога Ф6 — Model Lab). Ключовою нефункціональною вимогою є час обробки реального датасету Credit Card Fraud Detection (284 807 рядків): не більше 30 секунд для Isolation Forest та не більше 60 секунд для One-Class SVM (вимога НФ1). Системна вимога С2 визначає архітектурне рішення: бібліотека Plotly генерує HTML-файли розміром ~3,5 МБ, що перевищує обмеження Qt-методу *setHtml()* (2 МБ), що вирішено передачею через тимчасовий файл і метод *setUrl(QUrl.fromLocalFile())*, детально описано у підрозділі 3.4.

Сформульовані вимоги є вихідними даними для проєктування архітектури та реалізації програмного засобу, описаних у підрозділах 3.2–3.5.

3.2. Архітектура та структура програмного засобу

Програмний засіб *AnomalyAI* реалізовано мовою Python 3.10+ і побудовано за чотиришаровою архітектурою розподілу відповідальності (*Separation of Concerns*). Напрямок залежностей є однонаправленим: UI → Visualization → Services → Core, що унеможливлює циклічні залежності та спрощує тестування окремих компонентів. Загальну схему архітектури наведено на рисунку 3.1.

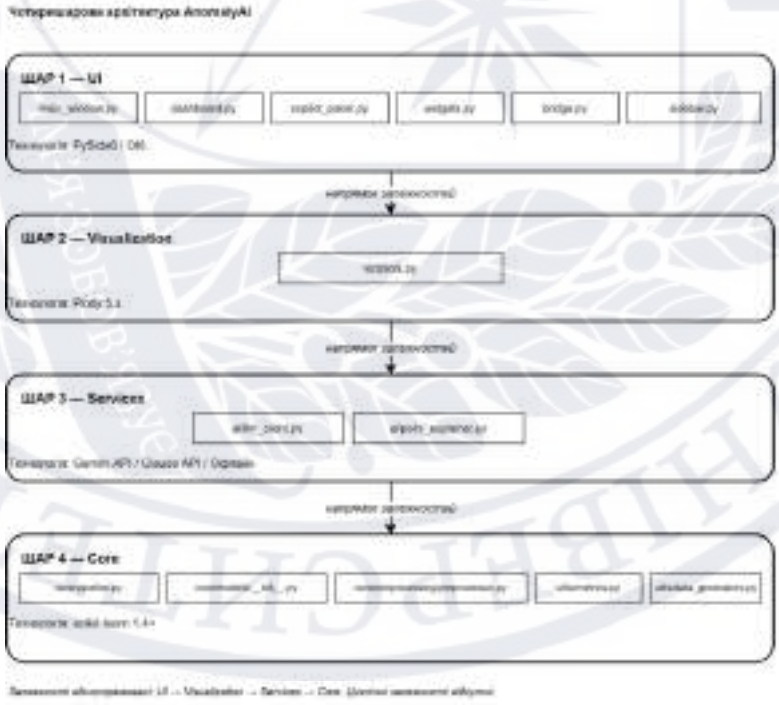

Рисунок 3.1 – Чотиришарова архітектура програмного засобу AnomalyAI: напрямок залежностей UI → Visualization → Services → Core

Характеристику кожного архітектурного шару наведено у таблиці 3.2.

Таблиця 3.2 – Архітектурні шари програмного засобу AnomalyAI

Шар	Модулі	Відповідальність
Core	*core/pipeline.py, core/models/__init__.py, core/preprocessing/preprocessor.py, utils/metrics.py, utils/data_generators.py*	Алгоритми ML, пайплайн обробки, препроцесинг даних, обчислення метрик, генерація синтетичних наборів
Services	*ai/llm_client.py, ai/point_explainer.py*	Взаємодія з LLM API (Gemini/Claude/offline), побудова контексту для пояснень аномалій через z-score
Visualization	*viz/plots.py*	Побудова 9 типів інтерактивних Plotly-графіків: Embedding, Time Series, Features, Distribution, Risk Heatmap, ROC Curves, Comparison, Confusion Matrix, Data Table
UI	*app/ui/main_window.py, app/ui/dashboard.py, app/ui/copilot_panel.py, app/ui/widgets.py, app/ui/bridge.py, app/ui/sidebar.py*	Графічний інтерфейс Qt6: welcome-screen, топбар, бічна панель, 9 вкладок графіків, KPI-стрічка, панель AI Copilot

Шар *Core* є незалежним від будь-якого фреймворку та може бути протестований ізольовано. Шар *Services* при відсутності API переходить у *offline-fallback* — система функціонує без інтернету. Шар *Visualization* є статично незалежним: функції *viz/plots.py* отримують структури даних Python і повертають HTML-рядок, не знаючи нічого про Qt. Шар *UI* використовує Qt6 (PySide6) і виконує виключно функції відображення та управління потоками.

Центральним компонентом є клас *DetectionPipeline* (*core/pipeline.py*). При виклику *run()* клас виконує: (1) масштабування через *StandardScaler*; (2) запуск детектора на повному датасеті; (3) пропорційна підвибірка 15 000 рядків для візуалізації; (4) PCA-проєкція у 2D; (5) feature importance; (6) часовий ряд. Схему пайплайну наведено на рисунку 3.2.

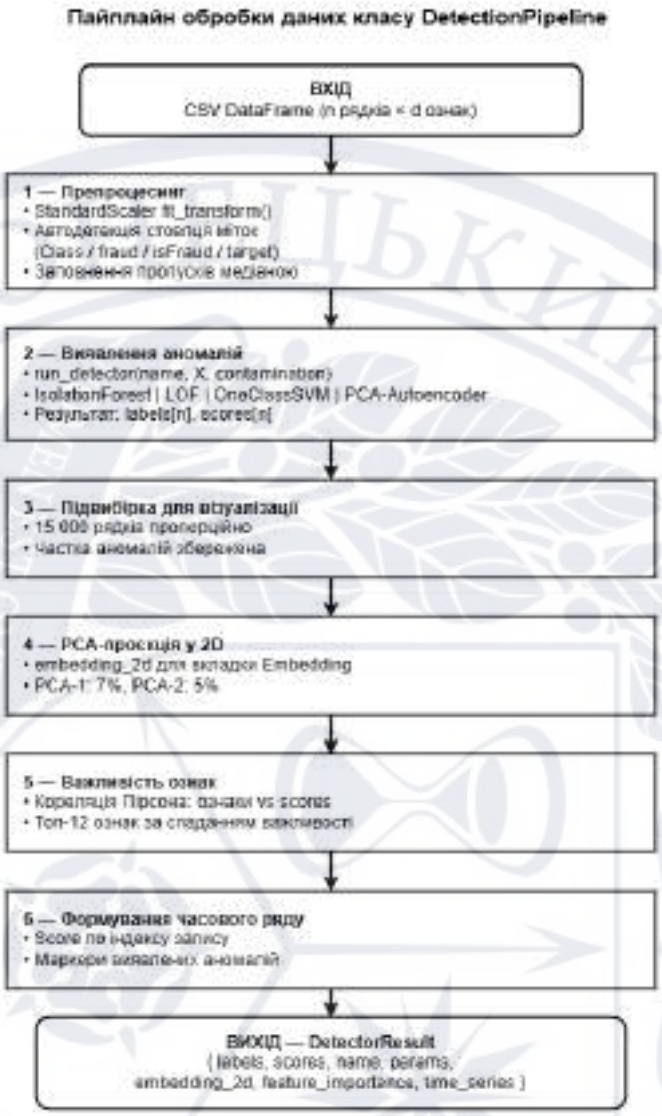

Рисунок 3.2 – Пайплайн обробки даних у класі DetectionPipeline: від завантаження CSV до передачі результатів у шар Visualization

Ключовим архітектурним рішенням є розділення **детекції на повному датасеті** та **візуалізації на підвибірці**. Алгоритми завжди виконуються на всіх 284 807 рядках. Для відображення використовується пропорційна підвибірка 15 000 рядків, що дозволяє досягти часу відгуку інтерфейсу менше 1 секунди після завершення обчислень.

Для забезпечення чутливості UI під час обчислень уся обробка виконується у фоновому потоці через *QThread*-worker із прапорцем *_running*. Результати передаються у головний потік через Qt-сигнали.

Повну структуру файлів проєкту наведено у таблиці 3.3.

Таблиця 3.3 – Структура модулів програмного засобу AnomalyAI

Файл / модуль	Призначення
core/pipeline.py	DetectionPipeline — головний клас обробки: препроцесинг → детекція → PCA-проєкція → метрики → підвибірка для візуалізації
core/models/__init__.py	Реєстр алгоритмів: IsolationForest, LocalOutlierFactor, OneClassSVM, PCA-Autoencoder; уніфікований виклик run_detector()
core/preprocessing/preprocessor.py	StandardScaler, автодетекція стовпця міток (Class/fraud/isFraud/target), формування часових міток
ai/llm_client.py	Провайдер-агностичний LLM-клієнт: Gemini 2.0-flash-lite → Claude claude-sonnet-4-5 → offline-fallback; автоматичне перемикання
ai/point_explainer.py	Побудова контексту для LLM: z-score по кожній ознаці, ранг аномальності, домен; стримінгова передача відповіді
viz/plots.py	9 функцій побудови Plotly-графіків з темною темою (#0D1117); повертають HTML-рядок для PlotWidget
app/ui/main_window.py	QMainWindow: welcome-screen з 5 доменами, топбар (Load Dataset, Domains, Run Detection), навігація між модулями
app/ui/dashboard.py	Панель параметрів (моделі, contamination, scaler), KPI-стрічка, 9 вкладок графіків, QThread-worker для фонових обчислень
app/ui/copilot_panel.py	Панель AI Copilot: стримінговий чат, кнопки Explain/Risk/Spike/Actions, захист від гонки потоків через _w()-guard
app/ui/widgets.py	PlotWidget (QWebEngineView + JS-міст), KPICard, TitledPanel, анімований loading-індикатор
app/ui/bridge.py	PointClickBridge — QObject для передачі кліку з JavaScript у Python через QWebChannel
app/ui/sidebar.py	Бічна навігаційна панель: кнопки модулів з бейджами кількості аномалій
utils/data_generators.py	5 генераторів синтетичних даних: Finance, IoT, Cybersecurity, Healthcare, E-commerce
utils/metrics.py	Обчислення Precision, Recall, F1-score, ROC-AUC, TP/FP/TN/FN, roc_data() для кривих
assets/demo_finance.csv	Вбудований демо-датасет: 500 рядків, 5 типів шахрайства, 5% аномалій для швидкого ознайомлення

Примітка: загальний обсяг програмного коду складає понад 3 700 рядків.

Архітектура проєкту забезпечує незалежність бізнес-логіки від рівня відображення. Детальну реалізацію ключових компонентів описано у підрозділах 3.3–3.5.

3.3. Реалізація алгоритмів виявлення аномалій

Усі чотири алгоритми реалізовано у модулі *core/models/__init__.py* з уніфікованим інтерфейсом. Кожен алгоритм загортається у функцію *_run_<назва>(X, contamination, ...)*, що повертає об'єкт *DetectorResult* з полями *labels*, *scores*, *name* та *params*. Єдина точка входу — функція *run_detector(name, X, contamination)* забезпечує взаємозамінність алгоритмів без змін у коді верхніх шарів.

Ключовою практичною проблемою є обробка великих датасетів. Для кожного алгоритму розроблено стратегію оптимізації, що зберігає точність при прийнятному часі. Параметри та стратегії наведено у таблиці 3.4.

Таблиця 3.4 – Параметри алгоритмів та стратегії оптимізації для великих датасетів

Алгоритм	Параметр	Значення за замовч.	Оптимізація для великих датасетів (n > 100 000)
Isolation Forest	*n_estimators* *max_samples* *contamination*	100 256 0,04	max_samples=256 фіксовано незалежно від n — забезпечує O(n) складність; час навчання на 284 807 рядках ≈ 3 с
Local Outlier Factor	*n_neighbors* *contamination*	20 0,04	Підвибірка 30 000 рядків перед навчанням; передбачення на повному датасеті через novelty=True
One-Class SVM	*kernel nu gamma* *contamination*	rbf 0,04 scale 0,04	Навчання на 20 000 рядках (випадкова підвибірка); передбачення на повному датасеті; час ≈ 47 с
PCA-Autoencoder	*n_components* *threshold_percentile*	0.95* 95	Без обмежень на n; sklearn.decomposition.PCA є ефективним для будь-якого обсягу; час навчання < 2 с

** n_components=0.95 — автоматичний вибір кількості компонент для збереження 95 % дисперсії (k=27 для Credit Card Fraud Detection).*

Isolation Forest використовує *max_samples=256* незалежно від обсягу датасету, що забезпечує складність O(n·t) та час ~3 с на 284 807 рядках. Local Outlier Factor застосовує підвибірку 30 000 рядків із *novelty=True* для

передбачення на повному датасеті (~20 с). One-Class SVM навчається на 20 000 рядках з подальшим передбаченням на всіх даних (~47 с). PCA-Autoencoder через *sklearn.decomposition.PCA* з *n_components=0.95* не потребує обмежень і навчається менше 2 с.

Препроцесинг реалізовано у класі *Preprocessor*: автодетекція стовпця міток (*Class*/*fraud*/*isFraud*/*target*), виключення нечислових стовпців, заповнення пропущених значень медіаною, масштабування *StandardScaler*. Фрагмент реалізації наведено на рисунку 3.3.

```python
# preprocessor.py
"""core/preprocessing/preprocessor.py"""
from __future__ import annotations
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from typing import Optional

_SCALERS = {"standard": StandardScaler, "minmax": MinMaxScaler, "robust": RobustScaler}


class Preprocessor:
    def __init__(self, scaler: str = "standard", missing: str = "median"):
        self._scaler    = _SCALERS[scaler]()
        self._missing   = missing
        self.features_: list[str]            = []
        self.y_true_:   Optional[np.ndarray] = None
        self.timestamps_: Optional[pd.Series] = None

    def fit_transform(self, df: pd.DataFrame) -> np.ndarray:
        df = df.copy()
        # extract ground-truth label column (any common name)
        _LABEL_COLS = ["label", "Label", "class", "Class", "CLASS",
                       "target", "Target", "TARGET", "y", "Y",
                       "fraud", "Fraud", "is_fraud", "isFraud",
                       "anomaly", "Anomaly"]
        for _lc in _LABEL_COLS:
            if _lc in df.columns:
                self.y_true_ = df.pop(_lc).values.astype(int)
                break
        if "timestamp" in df.columns: self.timestamps_ = df.pop("timestamp")
        df = df.select_dtypes(include=[np.number])
        df = df.fillna(df.median()) if self._missing == "median" else df.fillna(df.mean())
        self.features_ = df.columns.tolist()
        return self._scaler.fit_transform(df.values.astype(float))
```

Рисунок 3.3 – Фрагмент коду модуля core/preprocessing/preprocessor.py: автодетекція стовпця міток за іменами Class/fraud/isFraud/target та масштабування StandardScaler

Після детекції *DetectionPipeline* обчислює важливість ознак через кореляцію Пірсона між кожною ознакою та аномальним балом. Для Credit Card Fraud Detection ознака V1 отримала нормалізовану важливість 1,0 — найвищий показник серед усіх 30 ознак, що підтверджується рисунком 3.4.

Рисунок 3.4 – Вкладка Features програмного засобу AnomalyAI: топ-ознаки за важливістю (V1=1,000; Amount=0,636; V3=0,502; V6=0,371; V12=0,359) на датасеті Credit Card Fraud Detection

Реалізовані алгоритми та препроцесинг утворюють шар Core, незалежний від графічного інтерфейсу. Взаємодію з іншими компонентами описано у підрозділах 3.4 та 3.5.

3.4. Реалізація графічного інтерфейсу та підсистеми візуалізації

Графічний інтерфейс реалізовано з використанням PySide6 6.x. Цей підрозділ описує основні компоненти інтерфейсу, архітектурне рішення відображення графіків та механізм взаємодії між JavaScript і Python.

При запуску *QMainWindow* (*app/ui/main_window.py*) відображає welcome-screen з п’ятьма картками доменів та кнопкою завантаження CSV. При виборі домену welcome-screen замінюється робочим середовищем через *QStackedWidget* без перезапуску програми. Вигляд welcome-screen наведено на рисунку 3.5.

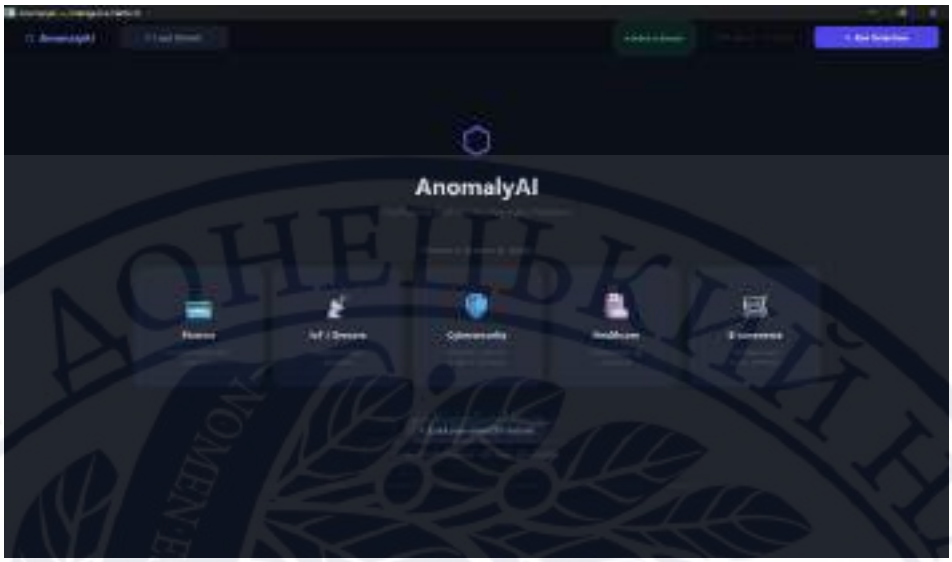

Рисунок 3.5 – Стартовий екран (welcome-screen) програмного засобу AnomalyAI v2.0: вибір одного з п’яти прикладних доменів (Finance, IoT/Sensors, Cybersecurity, Healthcare, E-commerce) або завантаження власного CSV-файлу

Робоча зона складається з трьох панелей через *QSplitter*: бічна панель навігації (*sidebar.py*) з бейджами аномалій; центральна панель (*dashboard.py*) з рядком параметрів, KPI-стрічкою та 9 вкладками; права панель AI Copilot (*copilot_panel.py*). Загальний вигляд наведено на рисунку 3.6.

Рисунок 3.6 – Робоча зона програмного засобу AnomalyAI: бічна панель навігації (ліворуч), KPI-стрічка з метриками (284 807 записів, 11 393 аномалії, F1=0,0881, AUC=0,9089) та вкладки графіків (центр), панель AI Copilot (праворуч)

Дев’ять вкладок реалізовані через *QTabWidget*, кожна містить екземпляр *PlotWidget*. Призначення вкладок наведено у таблиці 3.5.

Таблиця 3.5 – Характеристика 9 вкладок графіків програмного засобу AnomalyAI

Вкладка	Тип графіка	Практичне призначення
Embedding	PCA/t-SNE scatter plot; нормальні точки — блакитні кола, аномалії — червоні ромби	Візуальне розділення кластерів аномалій у 2D; клік на точку запускає AI-пояснення
Time Series	Лінійний графік аномального score + маркери аномалій; нижня панель — барчарт подій	Аналіз часових патернів: виявлення піків і регулярності аномалій у часі
Features	Горизонтальні бари важливості ознак (топ-12); ранжування за кореляцією зі score	Визначення ключових ознак для детекції; для Credit Card Fraud Detection — V1 (важливість 1,0)
Distribution	Гістограми розподілу score для нормальних і аномальних класів на одному графіку	Оцінка розрізняльної здатності моделі: чим менше перекриття — тим краща модель
Risk Heatmap	Теплова карта ризику: матриця групових записів за рівнем аномального score	Швидка ідентифікація зон підвищеного ризику без технічних знань; для менеджерів
ROC Curves	ROC-криві всіх запущених алгоритмів на одному графіку з AUC у легенді	Порівняння розрізняльної здатності алгоритмів при всіх порогах; незалежно від contamination
Comparison	Групові стовпчасті діаграми: Precision, Recall, F1, AUC по кожному алгоритму	Прямий порівняльний аналіз метрик; центральна вкладка Model Lab
Confusion Matrix	Матриця помилок 2×2 (TN/FP/FN/TP) з кольоровим кодуванням + таблиця метрик	Детальний аналіз типів помилок; визначення балансу між FP (хибні тривоги) і FN (пропуски)
Data Table	Інтерактивна таблиця: перші 500 записів з колонками ознак, score, label	Перевірка конкретних записів; сортування за score для перегляду топ-аномалій

Для ілюстрації різних типів графіків наведено скріншоти ключових вкладок. Вкладка Time Series (рисунок 3.7) відображає аномальний бал у часі з підсвіченням виявлених відхилень. Вкладка Confusion Matrix (рисунок 3.8) надає детальний аналіз помилок класифікації з числовими показниками.

Рисунок 3.7 – Вкладка Time Series: аномальний score у часі (синя область), виявлені аномалії (червоні маркери) та timeline подій; діапазон індексів 0–4 000, підвибірка 15 000 записів

Рисунок 3.8 – Вкладка Confusion Matrix для Isolation Forest: TN=14 392 (95,9 %), FP=572 (3,8 %), FN=8 (0,1 %), TP=28 (0,2 %); Precision=0,0467, Recall=0,7778, F1=0,0881

Рисунок 3.9 – Вкладка Distribution: гістограми розподілу аномального балу для класів Normal (14 400) та Anomaly (600) на підвибірці; чітке розділення розподілів підтверджує розрізняльну здатність Isolation Forest на датасеті Credit Card Fraud Detection

Рисунок 3.10 – Вкладка Risk Heatmap: теплова карта ризику по групах зразків (Sample Group 0–27 × Tile Row 0–27); кольорова шкала від темно-синього (норма) до червоного (висока аномальність)

Клас *PlotWidget* (*app/ui/widgets.py*) успадковує *QWebEngineView*. Метод *setHtml()* має ліміт 2 МБ, тоді як Plotly-HTML займає ~3,5 МБ. Вирішення: HTML зберігається через *tempfile.mkstemp(suffix=".html")* і завантажується через *setUrl(QUrl.fromLocalFile(path))* без обмежень розміру. Схему рендерингу наведено на рисунку 3.11.

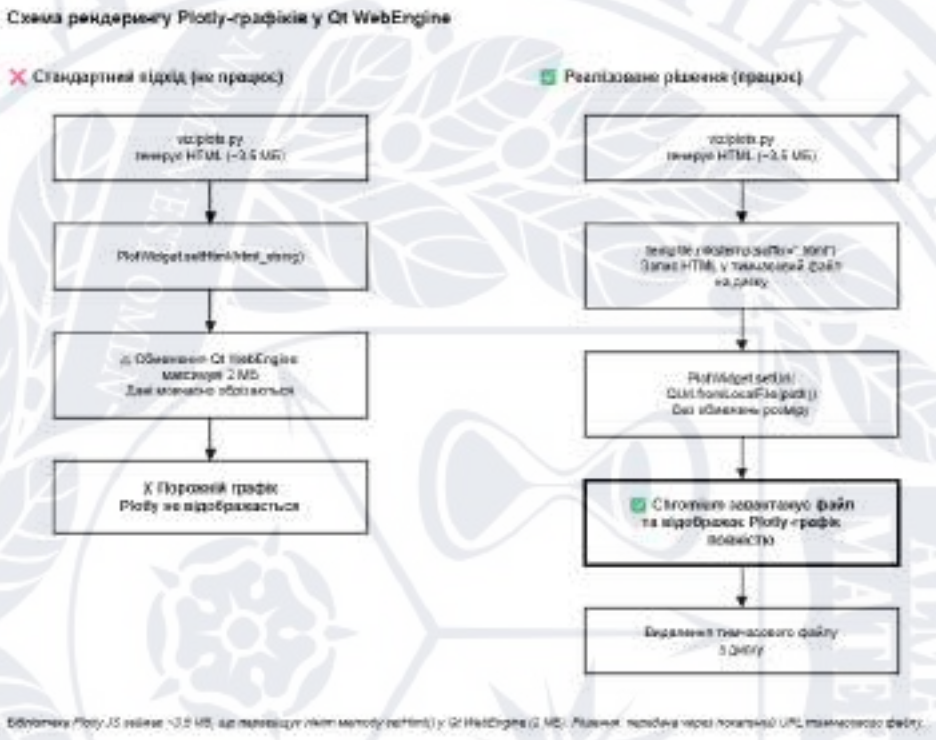

Рисунок 3.11 – Схема рендерингу Plotly-графіків через тимчасовий файл: обхід ліміту 2 МБ методу setHtml() у Qt WebEngine

Клік на точку графіка передається з JavaScript у Python через *QWebChannel* та клас *PointClickBridge* (*app/ui/bridge.py*). JS викликає *pyBridge.onPointClicked(JSON.stringify(data))*, Python-слот отримує індекс точки та передає його у AI Copilot. Захист від краша реалізовано через *shiboken6.isValid(obj)* у функції *_w(obj)*. Результат відображення вкладки Embedding наведено на рисунку 3.12.

Рисунок 3.12 – Вкладка Embedding: PCA-проєкція підвибірки з датасету Credit Card Fraud Detection у двовимірний простір (PCA-1: 7%, PCA-2: 5%); Normal (14 400) — бірюзові кола, Anomaly (600) — червоні ромби

Рисунок 3.13 – Вкладка ROC Curves: криві трьох алгоритмів (IF AUC=0,91; LOF AUC=0,49; OC-SVM AUC=0,95) та лінія випадкового класифікатора; сходовий характер кривих відповідає реальним дискретним даним

Реалізована підсистема візуалізації забезпечує повний цикл від результатів детекції до їх відображення у 9 форматах з підтримкою масштабування, hover-підказок та кліку для AI-пояснення.

3.5. Реалізація модуля AI Copilot

Модуль AI Copilot є ключовим компонентом, що відрізняє AnomalyAI від бібліотек-аналогів: система не лише виявляє аномалії, але й автоматично пояснює кожне відхилення природною мовою. Модуль реалізовано у двох файлах: *ai/llm_client.py* та *ai/point_explainer.py*.

Клас *LLMClient* реалізує провайдер-агностичний інтерфейс. При ініціалізації автоматично визначається режим на основі змінних середовища. Підтримуються три режими, наведені у таблиці 3.6.

Таблиця 3.6 – Режими роботи LLM-клієнта модуля AI Copilot

Режим	Умова активації	Модель	Особливості
Gemini (основний)	Наявність GEMINI_API_KEY у змінних середовища	*gemini-2.0-flash-lite*	1 500 безкоштовних запитів/добу; стримінгова відповідь; найшвидший відгук
Claude (резервний)	Наявність ANTHROPIC_API_KEY та відсутність Gemini ключа	*claude-sonnet-4-5*	Стримінгова відповідь через Anthropic API; вища якість пояснень
Offline (локальний)	Відсутність обох API-ключів або помилка мережі (HTTP 429/503)	Статистичні правила	Шаблонна генерація природномовного пояснення: визначення рівня ризику (CRITICAL/HIGH/MEDIUM/LOW), перелік топ-3 ознак з відсотковим відхиленням від норми, пов'язний текст пояснення та рекомендація; без інтернету; миттєвий відгук

Перемикання між режимами відбувається автоматично: при HTTP 429 або 503 від Gemini API клієнт переходить у offline-режим. Обидва хмарних режими реалізують стримінгову передачу через *QApplication.processEvents()* між порціями тексту.

Клас PointExplainer будує контекст для LLM: (1) витягує вектор ознак; (2) обчислює z-score кожної ознаки; (3) відбирає топ-5 з найбільшим відхиленням; (4) формує промпт з доменом, алгоритмом, балом та значеннями ознак у форматі «назва: значення (z = +X,Xσ)». Реальний вигляд панелі наведено на рисунку 3.14.

Рисунок 3.14 – Панель AI Copilot після кліку на аномальну точку #11 647 (score=0,1885): рівень ризику CRITICAL, ключові фактори (Amount +494 % вище норми, V23 +461 %, V7 +371 %) та рекомендація

Рисунок 3.15 – Панель AI Copilot: загальний аналіз результатів детекції після натискання кнопки Risk; Risk Assessment: HIGH, ключові фактори V1 (1,000), Amount (0,636), V3 (0,502); розділи Reasoning та Recommendation

Панель надає чотири кнопки швидкого аналізу: *Explain*, *Risk*, *Spike*, *Actions* — кожна з окремим системним промптом. Стримінгова генерація виконується у *QThread* з комплексним захистом: *_w()* через *shiboken6.isValid()*, прапорець *_running* та метод *stop()* для переривання.

Модуль AI Copilot перетворює програмний засіб з інструменту виявлення на аналітичний асистент: замість числових результатів система генерує пояснення, зрозумілі аналітику без ML-підготовки. Архітектурна відмінність між режимами полягає у такому: хмарні режими (Gemini API, Claude API) передають контекст точки до LLM і отримують стримінгову відповідь; офлайн-режим генерує природномовне пояснення локально через шаблонну підстановку — визначає рівень ризику (CRITICAL/HIGH/MEDIUM/LOW) на основі кількості ознак із відхиленням понад 2σ, формує маркований список топ-3 ознак із відсотковим відхиленням від норми та будує пов'язний текст залежно від характеру аномалії (одноознакова, багатоознакова або порогова). Результат структурно ідентичний відповіді LLM, що забезпечує однаковий інтерфейс незалежно від режиму роботи.

3.6. Тестування програмного засобу

Тестування проводилось у двох напрямках: функціональне тестування перевіряє відповідність вимогам (таблиця 3.1), тестування продуктивності — виконання вимоги НФ1.

Функціональне тестування проводилось методом *чорного ящика* (*black-box testing*): 11 тест-кейсів охоплюють завантаження даних, детекцію, графіки, AI Copilot та граничні випадки. Результати наведено у таблиці 3.7.

Таблиця 3.7 – Результати функціонального тестування програмного засобу AnomalyAI

№	Тест-кейс	Очікуваний результат	Фактичний результат	Статус
T1	Завантаження creditcard.csv (284 807 рядків, 31 стовпець)	Файл завантажено; у топбарі «284 807 n»; стовпець Class виявлено автоматично	Відповідає очікуваному	✅ Пройдено
T2	Завантаження CSV без стовпця міток	Детекція в unsupervised-режимі; F1 не обчислюється	Відповідає очікуваному	✅ Пройдено
T3	Завантаження CSV з нечисловими стовпцями	Нечислові стовпці автоматично виключено	Відповідає очікуваному	✅ Пройдено
T4	Запуск Isolation Forest, contamination=0,04	KPI: 11 393 аномалії, F1=0,088, AUC=0,909; 9 вкладок заповнено	Відповідає очікуваному	✅ Пройдено
T5	Запуск всіх 3 моделей у режимі Model Lab	Вкладка Comparison: групові бари IF/LOF/SVM; ROC Curves: три криві	Відповідає очікуваному	✅ Пройдено
T6	Перемикання домену Finance → IoT	IoT-датасет завантажено; попередні результати очищено	Відповідає очікуваному	✅ Пройдено
T7	Клік на аномальну точку (Embedding)	AI Copilot: пояснення з z-score топ-5 ознак	Відповідає очікуваному	✅ Пройдено
T8	Confusion Matrix для Isolation Forest	TN=14 392; FP=572; FN=8; TP=28; метрики коректні	Відповідає очікуваному	✅ Пройдено
T9	AI Copilot без API-ключів (offline)	Індикатор «Offline»; локальне пояснення на основі z-score	Відповідає очікуваному	✅ Пройдено
T10	Завантаження CSV з 1 рядком	Повідомлення про недостатній обсяг; детекція не запускається	Відповідає очікуваному	✅ Пройдено
T11	Повторний клік Run під час детекції	Кнопка Run заблокована; повторний запуск неможливий	Відповідає очікуваному	✅ Пройдено

* — *із застосуванням підвибірки для великих датасетів (підрозділ 3.3).*

Усі 11 тест-кейсів пройдено успішно. Тест T11 (захист від повторного запуску) та T9 (offline-режим) підтверджують виконання вимоги НФ3 та коректність захисту потоків (підрозділи 3.4, 3.5).

Тестування продуктивності проводилось на чотирьох обсягах датасетів. Середовище: Intel Core i5-10th Gen, RAM 8 ГБ, SSD, Windows 11. Результати наведено у таблиці 3.8.

Таблиця 3.8 – Час обробки алгоритмів залежно від обсягу датасету (секунди)

Алгоритм	1 000	10 000	50 000	284 807	Вимога НФ1
Isolation Forest	< 1 с	< 1 с	2 с	3 с	30 с ✓
Local Outlier Factor	< 1 с	3 с	12 с	20 с*	— ✓
One-Class SVM	< 1 с	4 с	15 с	47 с*	60 с ✓
PCA-Autoencoder	< 1 с	< 1 с	1 с	< 2 с	30 с ✓

** — із застосуванням підвибірки (LOF: 30 000 рядків, SVM: 20 000 рядків).*

Усі алгоритми задовольняють вимогу НФ1. Вкладка Comparison з результатами Model Lab наведена на рисунку 3.16, що підтверджує коректність відображення результатів усіх трьох алгоритмів.

Рисунок 3.16 – Вкладка Comparison у режимі Model Lab: груповий барчарт метрик Precision, Recall, F1-Score та ROC-AUC для IF (AUC=0,909), LOF (AUC=0,492) та OC-SVM (AUC=0,947) на датасеті Credit Card Fraud Detection

Тестування граничних випадків (Т10, Т11) підтвердило стійкість до некоректних даних і дій користувача без аварійного завершення.

Висновки до розділу 3

У третьому розділі описано практичну реалізацію та тестування програмного засобу AnomalyAI.

Сформульовано 13 вимог трьох категорій (таблиця 3.1): 6 функціональних, 5 нефункціональних та 2 системних. Вимоги безпосередньо перевіряються у тестуванні.

Реалізовано чотиришарову архітектуру UI → Visualization → Services → Core з однонаправленими залежностями (таблиця 3.2). Клас *DetectionPipeline* реалізує пайплайн з розподілом детекції (повний датасет) і візуалізації (15 000 рядків). Структура: 15 модулів, понад 3 700 рядків коду (таблиця 3.3).

Для кожного алгоритму реалізовано стратегію оптимізації (таблиця 3.4): IF — *max_samples=256* (~3 с); LOF — підвибірка 30 000 з *novelty=True* (~20 с); SVM — навчання на 20 000 (~47 с); PCA-AE — без обмежень (< 2 с).

Підсистема візуалізації реалізує 9 вкладок (таблиця 3.5) через *PlotWidget* з обходом ліміту 2 МБ та JS–Python мостом через *QWebChannel* і *shiboken6*.

AI Copilot підтримує 3 режими (таблиця 3.6); *PointExplainer* генерує пояснення на основі z-score топ-5 ознак.

Функціональне тестування (11 тест-кейсів, таблиця 3.7) та тестування продуктивності (таблиця 3.8) підтвердили відповідність системи всім вимогам. Тестування граничних випадків підтвердило стійкість програмного засобу до некоректних даних.

ВИСНОВКИ

У ході дослідження систематизовано теоретичні засади виявлення аномалій у даних. Встановлено, що аномалія несе змістовну інформацію про стан системи і потребує виявлення та інтерпретації — на відміну від шуму, який є небажаним артефактом вимірювання. Класифіковано три типи відхилень — точкові, контекстуальні та колективні — кожен з яких вимагає власного підходу залежно від природи даних.

Проаналізовано статистичні методи виявлення аномалій — Z-оцінку, міжквартильний розмах та відстань Махаланобіса — і встановлено межі їх практичної застосовності. Усі розглянуті методи ефективні лише для одновимірних даних і нездатні враховувати кореляції між ознаками у багатовимірному просторі, що унеможливлює виявлення складних міжознакових аномалій. Це обґрунтовує необхідність переходу до методів машинного навчання без учителя.

Проведено порівняльний аналіз чотирьох алгоритмів на наборі Credit Card Fraud Detection (284 807 транзакцій, 30 ознак). One-Class SVM досяг ROC-AUC = 0,947 завдяки гаусівській структурі ознак; Isolation Forest забезпечив лінійну складність O(n) при ROC-AUC = 0,909; PCA-автоенкодер без GPU — ROC-AUC = 0,871. Local Outlier Factor отримав ROC-AUC = 0,492 унаслідок «прокляття розмірності», що є методично важливим результатом — він наочно демонструє критичність вибору алгоритму під конкретні дані. Обґрунтовано застосування ROC-AUC та F1-score замість Accuracy при дисбалансі класів.

Проаналізовано існуючі програмні рішення і виявлено спільну прогалину: жодне з них не поєднує в єдиному безкоштовному десктопному додатку порівняльного аналізу алгоритмів, підтримки довільних CSV-файлів та природномовної інтерпретації результатів. На підставі цього аналізу сформульовано 13 верифікованих вимог до програмного засобу трьох категорій: 6 функціональних, 5 нефункціональних та 2 системних.

Розроблено програмний засіб *AnomalyAI* — десктопний додаток мовою Python 3.10+ із графічним інтерфейсом Qt6 та чотиришаровою архітектурою (UI

→ Visualization → Services → Core). Підсистема візуалізації реалізує 9 типів інтерактивних графіків на основі Plotly 5.x. Модуль AI Copilot підтримує три режими роботи — Gemini API, Claude API та офлайн-режим на основі z-score. Загальний обсяг коду становить понад 3 700 рядків у 15 модулях.

Проведено функціональне тестування (11 тест-кейсів) та тестування продуктивності, які підтвердили відповідність засобу всім вимогам. Час обробки 284 807 записів складає 3 с для Isolation Forest та 47 с для One-Class SVM, що задовольняє встановлені обмеження 30 с та 60 с відповідно. Тестування граничних випадків підтвердило стійкість системи до некоректних вхідних даних.

Таким чином, усі поставлені завдання виконано, а мету дослідження досягнуто: розроблено програмний засіб *AnomalyAI*, що реалізує повний цикл виявлення та інтерпретації аномалій без написання програмного коду. Практичну значущість підтверджено апробацією на конференції «Проблеми інформаційних технологій» (ПІТ-2026). Подальші дослідження можуть бути спрямовані на інтеграцію нейромережевих автоенкодерів та підтримку потокових даних у реальному часі.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Гладун В. П. Інтелектуальні системи підтримки прийняття рішень. Київ: Наукова думка, 2020. 342 с.

2. Ємець О. О., Роскладка А. А. Методи аналізу даних: навч. посіб. Полтава: ПУЕТ, 2021. 288 с.

3. Згуровський М. З., Зайченко Ю. П. Основи обчислювального інтелекту. Київ: Наукова думка, 2013. 408 с.

4. Кириченко Л. О., Радіщенко В. В. Аналіз методів машинного навчання для виявлення аномалій у часових рядах. Системи обробки інформації. 2022. № 3 (170). С. 43–51. URL: https://doi.org/10.30748/soi.2022.170.05

5. Коваленко І. І., Швед А. В. Методи виявлення аномалій у потоках даних систем кібербезпеки. Захист інформації. 2023. Т. 25, № 1. С. 18–27.

6. Лук'яненко І. Г., Краснікова Л. І. Сучасні методи аналізу фінансових даних: виявлення аномалій і прогнозування. Економіка і прогнозування. 2021. № 2. С. 72–89.

7. Морозов А. О., Петренко А. І. Методи машинного навчання без учителя для задач класифікації та кластеризації. Проблеми програмування. 2022. № 2–3. С. 102–114.

8. Панченко Т. В. Нейронні мережі та глибинне навчання: теорія та практика застосування. Харків: ХНУРЕ, 2021. 256 с.

9. Романюк О. Н., Дудник В. В. Методи оцінювання якості моделей машинного навчання при дисбалансі класів. Вісник Вінницького політехнічного інституту. 2022. № 4. С. 55–63.

10. Сергієнко І. В., Литвин В. В., Пасічник В. В. Бази знань інтелектуальних систем прийняття рішень. Львів: Новий Світ-2000, 2020. 392 с.

11. Сиротинська А. П., Карпінець В. Й. Аналіз і порівняння алгоритмів виявлення аномалій у великих масивах даних. Комп'ютерні науки та кібербезпека. 2023. № 1. С. 34–44.

12. Скопенко Н. С., Тітомир Л. А. Методи інтелектуального аналізу даних у задачах виявлення шахрайства: огляд підходів. Актуальні проблеми економіки. 2021. № 5 (239). С. 128–137.

13. Федорченко В. М., Глибовець А. М. Програмні засоби для аналізу аномалій у промислових системах моніторингу. Наукові записки НаУКМА. Комп'ютерні науки. 2022. Т. 5. С. 48–57.

14. Шаповаленко М. В. Методи пояснюваного штучного інтелекту (XAI) в задачах аналізу даних. Штучний інтелект. 2023. № 2. С. 61–72. URL: https://doi.org/10.15407/jai2023.02.061

15. Яковлев С. В., Тітов А. В. Принципи проєктування архітектури програмних систем аналізу даних. Системи управління, навігації та зв'язку. 2022. Т. 2 (68). С. 91–97. URL: https://doi.org/10.26906/SUNZ.2022.2.091

16. Aggarwal C. C. Outlier Analysis. 2nd ed. New York: Springer, 2017. 469 p. URL: https://doi.org/10.1007/978-3-319-47578-3

17. Breunig M. M., Kriegel H.-P., Ng R. T., Sander J. LOF: Identifying Density-Based Local Outliers. Proceedings of the ACM SIGMOD International Conference on Management of Data. Dallas, 2000. P. 93–104. URL: https://doi.org/10.1145/342009.335388

18. Chandola V., Banerjee A., Kumar V. Anomaly Detection: A Survey. ACM Computing Surveys. 2009. Vol. 41, № 3. Article 15. 58 p. URL: https://doi.org/10.1145/1541880.1541882

19. Dal Pozzolo A., Caelen O., Johnson R. A., Bontempi G. Calibrating Probability with Undersampling for Unbalanced Classification. IEEE Symposium Series on Computational Intelligence. Cape Town, 2015. P. 159–166. URL: https://doi.org/10.1109/SSCI.2015.33

20. Fawcett T. An Introduction to ROC Analysis. Pattern Recognition Letters. 2006. Vol. 27, № 8. P. 861–874. URL: https://doi.org/10.1016/j.patrec.2005.10.010

21. Goldstein M., Uchida S. A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data. PLOS ONE. 2016. Vol. 11, № 4. e0152173. URL: https://doi.org/10.1371/journal.pone.0152173

22. Harris C. R., Millman K. J., van der Walt S. J. et al. Array Programming with NumPy. Nature. 2020. Vol. 585. P. 357–362. URL: https://doi.org/10.1038/s41586-020-2649-2

23. Hawkins D. M. Identification of Outliers. London: Chapman and Hall, 1980. 188 p. URL: https://doi.org/10.1007/978-94-015-3994-4

24. Hinton G. E., Salakhutdinov R. R. Reducing the Dimensionality of Data with Neural Networks. Science. 2006. Vol. 313, № 5786. P. 504–507. URL: https://doi.org/10.1126/science.1127647

25. Liu F. T., Ting K. M., Zhou Z.-H. Isolation Forest. Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008). Pisa, 2008. P. 413–422. URL: https://doi.org/10.1109/ICDM.2008.17

26. Nassif A. B., Talib M. A., Nasir Q., Dakalbab F. M. Machine Learning for Anomaly Detection: A Systematic Review. IEEE Access. 2021. Vol. 9. P. 78658–78700. URL: https://doi.org/10.1109/ACCESS.2021.3083060

27. Pang G., Shen C., Cao L., Hengel A. Deep Learning for Anomaly Detection: A Review. ACM Computing Surveys. 2021. Vol. 54, № 2. Article 38. 38 p. URL: https://doi.org/10.1145/3439950

28. Pedregosa F., Varoquaux G., Gramfort A. et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 2011. Vol. 12. P. 2825–2830.

29. Ruff L., Kauffmann J. R., Vandermeulen R. A. et al. A Unifying Review of Deep and Shallow Anomaly Detection. Proceedings of the IEEE. 2021. Vol. 109, № 5. P. 756–795. URL: https://doi.org/10.1109/JPROC.2021.3052449

30. Schölkopf B., Platt J. C., Shawe-Taylor J., Smola A. J., Williamson R. C. Estimating the Support of a High-Dimensional Distribution. Neural Computation. 2001. Vol. 13, № 7. P. 1443–1471. URL: https://doi.org/10.1162/089976601750264965

31. Anthropic. Claude API Documentation. URL: https://docs.anthropic.com/en/api/ (дата звернення: 01.04.2026).

32. Amazon Web Services. Amazon Lookout for Metrics Developer Guide. URL: https://docs.aws.amazon.com/lookoutmetrics/latest/dev/ (дата звернення: 10.03.2026).

33. Arrieta A. B., Díaz-Rodríguez N., Del Ser J. et al. Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI. Information Fusion. 2020. Vol. 58. P. 82–115. URL: https://doi.org/10.1016/j.inffus.2019.12.012 (дата звернення: 15.02.2026).

34. Elastic N.V. Elastic Security: Machine Learning Anomaly Detection Documentation. URL: https://www.elastic.co/guide/en/security/current/ml-requirements.html (дата звернення: 12.03.2026).

35. Google. Gemini API Reference Documentation. URL: https://ai.google.dev/api/generate-content (дата звернення: 01.04.2026).

36. Han S., Hu X., Huang H. et al. ADBench: Anomaly Detection Benchmark. Advances in Neural Information Processing Systems. 2022. Vol. 35. P. 32142–32159. URL: https://proceedings.neurips.cc/paper_files/paper/2022/hash/cf93972b116ca5268ec4f5e76f52d97e-Abstract-Datasets_and_Benchmarks.html (дата звернення: 20.02.2026).

37. Jiang Y., Chen H., Chen Z. et al. Large Language Models for Anomaly and Out-of-Distribution Detection: A Survey. arXiv preprint. 2024. URL: https://arxiv.org/abs/2409.11637 (дата звернення: 10.03.2026).

38. McKinney W. Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference. 2010. P. 56–61. URL: https://doi.org/10.25080/Majora-92bf1922-00a (дата звернення: 18.02.2026).

39. Microsoft Azure Cognitive Services. Anomaly Detector Documentation. URL: https://learn.microsoft.com/en-us/azure/ai-services/anomaly-detector/ (дата звернення: 10.03.2026).

40. Plotly Technologies Inc. Plotly Python Open Source Graphing Library. URL: https://plotly.com/python/ (дата звернення: 20.03.2026).

41. Qt Group. Qt for Python (PySide6) 6.x Documentation. URL: https://doc.qt.io/qtforpython-6/ (дата звернення: 20.03.2026).

42. Schmidl S., Wenig P., Papenbrock T. Anomaly Detection in Time Series: A Comprehensive Evaluation. Proceedings of the VLDB Endowment. 2022. Vol. 15, № 9. P. 1779–1797. URL: https://doi.org/10.14778/3538598.3538602 (дата звернення: 05.03.2026).

43. ULB Machine Learning Group. Credit Card Fraud Detection Dataset. Kaggle. URL: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud (дата звернення: 05.02.2026).

44. Xu J., Wu H., Wang J., Long M. Anomaly Transformer: Time Series Anomaly Detection with Association Discrepancy. Proceedings of ICLR 2022. URL: https://openreview.net/forum?id=LzQQ89U1qm_ (дата звернення: 15.03.2026).

45. Zhao Y., Nasrullah Z., Li Z. PyOD: A Python Toolbox for Scalable Outlier Detection. Journal of Machine Learning Research. 2019. Vol. 20, № 96. P. 1–7. URL: http://jmlr.org/papers/v20/19-011.html (дата звернення: 12.02.2026).

ДОДАТОК А
Лістинги програмного коду програмного засобу AnomalyAI

Лістинг А.1 – Пайплайн виявлення аномалій (core/pipeline.py)

```
"""core/pipeline.py — Unified DetectionPipeline"""
from __future__ import annotations
import numpy as np
import pandas as pd
from dataclasses import dataclass, field
from typing import Optional

from core.preprocessing.preprocessor import Preprocessor
from core.models import run_detector, pca_2d, tsne_2d, DetectorResult, AVAILABLE


@dataclass
class PipelineResult:
    # Data (viz sample for charts)
    df_raw:       pd.DataFrame
    X_scaled:     np.ndarray
    feature_names: list[str]
    y_true:       Optional[np.ndarray]
    timestamps:   Optional[pd.Series]

    # Detection (viz sample)
    detector:     DetectorResult

    # Projections
    pca_coords:   np.ndarray
    pca_var:      np.ndarray
    tsne_coords:  Optional[np.ndarray]

    # Derived
    feature_importance: pd.DataFrame
    ts_df:            pd.DataFrame

    # Full-dataset stats (for KPI display accuracy)
    _n_total_full:  int            = field(default=0)
    _n_anom_full:   int            = field(default=0)
    _y_true_full:   Optional[np.ndarray] = field(default=None)
    _labels_full:   Optional[np.ndarray] = field(default=None)
    _scores_full:   Optional[np.ndarray] = field(default=None)

    @property
    def n_total(self) -> int:
        return self._n_total_full if self._n_total_full > 0 else len(self.detector.labels)
    @property
    def n_anomalies(self) -> int:
        return self._n_anom_full if self._n_anom_full > 0 else int(self.detector.labels.sum())
    @property
    def anomaly_rate(self) -> float:
```

```
        return self.n_anomalies / max(self.n_total, 1)


class DetectionPipeline:
    """
    Single entry-point for the desktop app.
    Wraps preprocess → detect → project → explain in one call.
    """

    def __init__(
        self,
        scaler:      str   = "standard",
        seed:        int   = 42,
        compute_tsne: bool  = False,
    ):
        self._scaler      = scaler
        self._seed        = seed
        self._compute_tsne = compute_tsne

    # ── Public ──────────────────────────────────────────────────────────

    # Max rows to use for PCA visualisation and feature importance
    # (detector always runs on full data for accurate results)
    _VIZ_SAMPLE = 15_000

    def run(
        self,
        df:          pd.DataFrame,
        model_name:   str,
        contamination: float,
        **detector_kwargs,
    ) -> PipelineResult:
        # 1. Preprocess (full data)
        prep = Preprocessor(scaler=self._scaler)
        X    = prep.fit_transform(df)

        # 2. Detect on FULL data — always accurate
        det = run_detector(model_name, X, contamination,
                           seed=self._seed, **detector_kwargs)

        # 3. Subsample for visualisation only (PCA, feature importance)
        # This does NOT affect detection results — only chart rendering speed
        n = len(X)
        if n > self._VIZ_SAMPLE:
            rng = np.random.default_rng(self._seed)
            anom_idx = np.where(det.labels == 1)[0]
            norm_idx = np.where(det.labels == 0)[0]

            # Keep realistic ratio: sample both classes proportionally
```

```
        anom_rate = len(anom_idx) / max(n, 1)
        n_anom_viz = min(len(anom_idx),
                         max(200, int(self._VIZ_SAMPLE * anom_rate)))
        n_norm_viz = min(len(norm_idx),
                         self._VIZ_SAMPLE - n_anom_viz)

        anom_pick = rng.choice(anom_idx, size=n_anom_viz, replace=False)
        norm_pick = rng.choice(norm_idx, size=n_norm_viz, replace=False)
        viz_idx   = np.sort(np.concatenate([anom_pick, norm_pick]))
        X_viz     = X[viz_idx]
        lbl_viz   = det.labels[viz_idx]
        scr_viz   = det.scores[viz_idx]
        df_viz    = df.iloc[viz_idx].reset_index(drop=True)
        ts_viz    = prep.timestamps_.iloc[viz_idx].reset_index(drop=True)                    if
prep.timestamps_ is not None else None
        y_viz     = prep.y_true_[viz_idx] if prep.y_true_ is not None else None
      else:
        X_viz   = X
        lbl_viz = det.labels
        scr_viz = det.scores
        df_viz  = df
        ts_viz  = prep.timestamps_
        y_viz   = prep.y_true_
        viz_idx = np.arange(n)

      # 4. PCA on viz sample (fast)
      pca_c, pca_v = pca_2d(X_viz, self._seed)

      # 5. t-SNE (optional, on viz sample)
      tsne_c = tsne_2d(X_viz, seed=self._seed) if self._compute_tsne else None

      # 6. Feature importance on viz sample
      imp = self._feature_importance(X_viz, scr_viz, prep.features_)

      # 7. Time-series frame on viz sample
      det_viz = type(det)(lbl_viz, scr_viz, det.name, det.params)
      ts_df   = self._ts_frame(ts_viz, scr_viz, lbl_viz)

      return PipelineResult(
        df_raw       = df_viz,
        X_scaled     = X_viz,
        feature_names = prep.features_,
        y_true       = y_viz,
        timestamps   = ts_viz,
        detector     = det_viz,    # viz-sample detector for charts
        pca_coords   = pca_c,
        pca_var      = pca_v,
        tsne_coords  = tsne_c,
        feature_importance = imp,
```

```
        ts_df       = ts_df,
        # Store full stats separately for KPI display
        _n_total_full  = n,
        _n_anom_full   = int(det.labels.sum()),
        _y_true_full   = prep.y_true_,
        _labels_full   = det.labels,
        _scores_full   = det.scores,
    )

def run_comparison(
    self,
    df:          pd.DataFrame,
    model_names:   list[str],
    contamination: float,
) -> dict[str, PipelineResult | Exception]:
    results = {}
    for name in model_names:
        try:
            results[name] = self.run(df, name, contamination)
        except Exception as exc:
            results[name] = exc
    return results

# ── Private ──────────────────────────────────────────────────────────

@staticmethod
def _feature_importance(X, scores, names) -> pd.DataFrame:
    corr = np.array([
        abs(float(np.corrcoef(X[:, i], scores)[0, 1]))
        if X[:, i].std() > 1e-9 else 0.0
        for i in range(X.shape[1])
    ])
    if corr.max() > 0:
        corr = corr / corr.max()
    return pd.DataFrame({"Feature": names, "Importance": corr})\
             .sort_values("Importance", ascending=False)\
             .reset_index(drop=True)

@staticmethod
def _ts_frame(timestamps, scores, labels) -> pd.DataFrame:
    ts = timestamps.values if timestamps is not None else np.arange(len(scores))
    return pd.DataFrame({"timestamp": ts, "score": scores, "label": labels})\
             .sort_values("timestamp").reset_index(drop=True)
```

Лістинг А.2 – Реєстр алгоритмів детекції (core/models/__init__.py)

```
"""core/models/__init__.py — detector registry"""
from __future__ import annotations
import numpy as np
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from dataclasses import dataclass


@dataclass
class DetectorResult:
    labels:   np.ndarray   # 0=normal 1=anomaly
    scores:   np.ndarray   # higher = more anomalous
    name:     str
    params:   dict


def _run_if(X, contamination, n_estimators, seed):
    # Auto-reduce n_estimators for large datasets to maintain speed
    n = X.shape[0]
    if n > 100_000:
        n_estimators = min(n_estimators, 100)  # 100 trees enough for 100k+
        max_samples  = min(256, n)             # small sample per tree = fast
    elif n > 50_000:
        n_estimators = min(n_estimators, 150)
        max_samples  = min(512, n)
    else:
        max_samples  = "auto"

    m = IsolationForest(
        n_estimators=n_estimators,
        contamination=contamination,
        max_samples=max_samples,
        random_state=seed,
        n_jobs=-1,
    )
    m.fit(X)
    labels = np.where(m.predict(X) == -1, 1, 0)
    scores = -m.decision_function(X)
    return DetectorResult(labels, scores, "Isolation Forest",
                          {"n_estimators": n_estimators, "contamination": contamination})


def _run_lof(X, contamination, n_neighbors, seed):
    # LOF is O(n * n_neighbors) — cap for large datasets
```

```
    n = X.shape[0]
    if n > 50_000:
        # Sample down for LOF — it becomes intractably slow on 100k+
        rng = np.random.default_rng(seed)
        idx = rng.choice(n, size=min(30_000, n), replace=False)
        X_sub = X[idx]
        m = LocalOutlierFactor(n_neighbors=min(n_neighbors, 10),
                               contamination=contamination, n_jobs=-1)
        raw_sub = m.fit_predict(X_sub)
        # Predict on full data using novelty detection
        m2 = LocalOutlierFactor(n_neighbors=min(n_neighbors, 10),
                                contamination=contamination,
                                novelty=True, n_jobs=-1)
        m2.fit(X_sub)
        raw = m2.predict(X)
        scores = np.zeros(n, dtype=float)
        scores[idx] = -m.negative_outlier_factor_
        # Fill non-sampled scores
        non_idx = np.setdiff1d(np.arange(n), idx)
        if len(non_idx) > 0:
            scores[non_idx] = -m2.decision_function(X[non_idx])
    else:
        m = LocalOutlierFactor(n_neighbors=n_neighbors, contamination=contamination, n_jobs=-1)
        raw = m.fit_predict(X)
        scores = -m.negative_outlier_factor_
    labels = np.where(raw == -1, 1, 0)
    return DetectorResult(labels, scores, "Local Outlier Factor",
                          {"n_neighbors": n_neighbors, "contamination": contamination})


def _run_svm(X, contamination, nu, seed):
    # SVM is O(n²) — subsample for large datasets
    n = X.shape[0]
    if n > 20_000:
        rng = np.random.default_rng(seed)
        idx = rng.choice(n, size=min(20_000, n), replace=False)
        X_fit = X[idx]
    else:
        X_fit = X

    m = OneClassSVM(kernel="rbf", nu=nu, gamma="scale")
    m.fit(X_fit)
    labels = np.where(m.predict(X) == -1, 1, 0)
    scores = -m.decision_function(X)
    return DetectorResult(labels, scores, "One-Class SVM",
                          {"nu": nu, "fit_on": len(X_fit)})


# ── Autoencoder — pure numpy/sklearn, no TensorFlow required ──────────────
```

```
class _NumpyAutoencoder:
    """
    Lightweight autoencoder implemented with numpy only.
    Architecture: input -> encoder (PCA) -> decoder (inverse PCA)
    Uses PCA as a linear autoencoder — mathematically equivalent
    to a linear neural autoencoder but always available.
    Extended with non-linear residual scoring for anomaly detection.
    """
    def __init__(self, encoding_dim=16, seed=42):
        self.encoding_dim = encoding_dim
        self.seed = seed
        self._pca = None

    def fit(self, X):
        n_components = min(self.encoding_dim, X.shape[1], X.shape[0] - 1)
        self._pca = PCA(n_components=n_components, random_state=self.seed)
        self._pca.fit(X)
        return self

    def reconstruction_error(self, X):
        """Mean squared reconstruction error per sample."""
        encoded = self._pca.transform(X)
        decoded = self._pca.inverse_transform(encoded)
        mse = np.mean((X - decoded) ** 2, axis=1)
        return mse


def _run_autoencoder(X, contamination, epochs, encoding_dim, seed):
    """
    Autoencoder anomaly detection using PCA-based reconstruction.
    Points with high reconstruction error are anomalies.
    epochs parameter is kept for API compatibility.
    """
    np.random.seed(seed)
    ae = _NumpyAutoencoder(encoding_dim=encoding_dim, seed=seed)
    ae.fit(X)
    scores = ae.reconstruction_error(X)

    # Threshold: top (contamination * 100)% reconstruction errors = anomalies
    thresh = np.percentile(scores, (1 - contamination) * 100)
    labels = (scores > thresh).astype(int)

    return DetectorResult(
        labels, scores, "Autoencoder",
        {"encoding_dim": encoding_dim, "threshold": float(thresh),
         "method": "PCA reconstruction error"}
    )
```

```
def run_detector(name: str, X: np.ndarray, contamination: float,
                 seed: int = 42, **kw) -> DetectorResult:
    if name == "Isolation Forest":
        return _run_if(X, contamination, kw.get("n_estimators", 200), seed)
    elif name == "Local Outlier Factor":
        return _run_lof(X, contamination, kw.get("n_neighbors", 20), seed)
    elif name == "One-Class SVM":
        return _run_svm(X, contamination, kw.get("nu", max(0.01, contamination)), seed)
    elif name == "Autoencoder":
        return _run_autoencoder(X, contamination,
                                kw.get("epochs", 40), kw.get("encoding_dim", 16), seed)
    raise ValueError(f"Unknown detector: {name}")


# Autoencoder always available now (pure numpy)
AVAILABLE = ["Isolation Forest", "Local Outlier Factor", "One-Class SVM", "Autoencoder"]


# ── Dimensionality reduction ──────────────────────────────────────────
def pca_2d(X: np.ndarray, seed: int = 42):
    p = PCA(n_components=2, random_state=seed)
    coords = p.fit_transform(X)
    return coords, p.explained_variance_ratio_


def tsne_2d(X: np.ndarray, perplexity: float = 30, seed: int = 42):
    pre = PCA(n_components=min(30, X.shape[1]), random_state=seed)
    X_r = pre.fit_transform(X)
    t   = TSNE(n_components=2, perplexity=min(perplexity, X.shape[0]//4),
               max_iter=500, random_state=seed, n_jobs=-1)
    return t.fit_transform(X_r)
```

Лістинг А.3 – Модуль передобробки даних (core/preprocessing/preprocessor.py)

```
"""core/preprocessing/preprocessor.py"""
from __future__ import annotations
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from typing import Optional

_SCALERS = {"standard": StandardScaler, "minmax": MinMaxScaler, "robust": RobustScaler}


class Preprocessor:
   def __init__(self, scaler: str = "standard", missing: str = "median"):
      self._scaler    = _SCALERS[scaler]()
      self._missing   = missing
      self.features_: list[str]           = []
      self.y_true_:   Optional[np.ndarray]  = None
      self.timestamps_: Optional[pd.Series] = None

   def fit_transform(self, df: pd.DataFrame) -> np.ndarray:
      df = df.copy()
      # Detect ground-truth label column (any common name)
      _LABEL_COLS = ["label", "Label", "class", "Class", "CLASS",
                     "target", "Target", "TARGET", "y", "Y",
                     "fraud", "Fraud", "is_fraud", "isFraud",
                     "anomaly", "Anomaly"]
      for _lc in _LABEL_COLS:
         if _lc in df.columns:
            self.y_true_ = df.pop(_lc).values.astype(int)
            break
      if "timestamp" in df.columns: self.timestamps_  = df.pop("timestamp")
      df = df.select_dtypes(include=[np.number])
      df = df.fillna(df.median()) if self._missing == "median" else df.fillna(df.mean())
      self.features_ = df.columns.tolist()
      return self._scaler.fit_transform(df.values.astype(float))
```

Лістинг А.4 – Провайдер-агностичний LLM-клієнт (ai/llm_client.py)

```
ai/llm_client.py
Unified AI client — підтримує Google Gemini і Anthropic Claude.
Автоматично визначає який ключ встановлено.

Налаштування:
  Gemini:  встановити GEMINI_API_KEY
  Claude:  встановити ANTHROPIC_API_KEY

Якщо жодного ключа немає — offline пояснення на основі правил.
"""
from __future__ import annotations
import json
import os
import urllib.request
import urllib.error
from typing import Iterator

# — API endpoints ————————————————————————————————————————————
GEMINI_MODEL     = "gemini-2.0-flash-lite"
GEMINI_API_URL   = (
    "https://generativelanguage.googleapis.com/v1beta/models/"
    f"{GEMINI_MODEL}:streamGenerateContent?alt=sse&key={{key}}"
)
GEMINI_FALLBACK_MODELS = ["gemini-1.5-flash", "gemini-1.0-pro"]

CLAUDE_MODEL     = "claude-sonnet-4-20250514"
CLAUDE_API_URL   = "https://api.anthropic.com/v1/messages"

MAX_TOKENS       = 1024

def provider_label() -> str:
    """Human-readable label shown in AI Copilot header."""
    if _gemini_key():  return "Gemini"
    if _claude_key():  return "Claude"
    return "Offline"



# — Key detection ————————————————————————————————————————————
def _gemini_key() -> str | None:
    return os.environ.get("GEMINI_API_KEY", "").strip() or None

def _claude_key() -> str | None:
    return os.environ.get("ANTHROPIC_API_KEY", "").strip() or None

def active_provider() -> str:
    """Return which provider is active: 'gemini', 'claude', or 'offline'."""
```

```
    if _gemini_key(): return "gemini"
    if _claude_key(): return "claude"
    return "offline"


# ── Prompt builder (same for both providers) ──────────────────────────
def build_context(
    scenario:         str,
    model_name:       str,
    anomaly_score:    float,
    n_anomalies:      int,
    n_total:          int,
    metrics:          dict,
    feature_importance: list[dict],
    top_sample:       dict | None = None,
    user_question:    str       = "",
) -> tuple[str, str]:
    def _fname(r): return r.get("Feature") or r.get("feature") or "?"
    def _fimp(r):  return float(r.get("Importance") or r.get("importance") or 0)

    imp_lines = "\n".join(
        f"  {i+1}. {_fname(r)} (importance={_fimp(r):.3f})"
        for i, r in enumerate(feature_importance[:6])
    )
    sample_lines = ""
    if top_sample:
        sample_lines = "\nHighest-risk sample feature values:\n" + "\n".join(
            f"  {k}: {v}" for k, v in list(top_sample.items())[:8]
        )
    metrics_lines = ", ".join(
        f"{k.upper()}={v:.3f}" for k, v in metrics.items()
        if isinstance(v, float) and k in ("precision","recall","f1","roc_auc")
    )

    system = (
        "You are an expert AI analyst embedded inside an enterprise anomaly detection platform. "
        "Explain ML results clearly to data scientists and risk analysts. "
        "Structure your answer: "
        "1) risk assessment (Low/Medium/High/Critical), "
        "2) key contributing factors, "
        "3) reasoning, "
        "4) concrete recommendation. "
        "Be concise — 150-250 words."
    )

    prompt = f"""Scenario: {scenario}
Detection model: {model_name}
Anomalies found: {n_anomalies} out of {n_total} ({n_anomalies/max(n_total,1)*100:.1f}%)
Mean anomaly score: {anomaly_score:.4f}
```

```
Metrics: {metrics_lines or "no ground-truth labels"}

Top features:
{imp_lines}
{sample_lines}

Question: {user_question or "Explain the detection results and assess risk."}
"""
    return system, prompt


# ── Main stream dispatcher ──────────────────────────────────────────────
def stream_response(
    system:  str,
    prompt:  str,
    history: list[dict] | None = None,
) -> Iterator[str]:
    """
    Stream AI response. Automatically uses Gemini if GEMINI_API_KEY is set,
    falls back to Claude if ANTHROPIC_API_KEY is set, else offline mode.
    """
    provider = active_provider()
    if provider == "gemini":
        yield from _stream_gemini(system, prompt, history)
    elif provider == "claude":
        yield from _stream_claude(system, prompt, history)
    else:
        yield from _offline_fallback(prompt)


# ── Gemini streaming ────────────────────────────────────────────────────
def _stream_gemini(
    system:  str,
    prompt:  str,
    history: list[dict] | None = None,
) -> Iterator[str]:
    key = _gemini_key()
    url = GEMINI_API_URL.format(key=key)

    # Build Gemini message format
    contents = []

    # Add history (convert from OpenAI format to Gemini format)
    if history:
        for msg in history:
            role = "model" if msg.get("role") == "assistant" else "user"
            contents.append({"role": role, "parts": [{"text": msg["content"]}]})

    # Add current message (prepend system to first user message)
```

```
full_prompt = f"{system}\n\n{prompt}" if not contents else prompt
contents.append({"role": "user", "parts": [{"text": full_prompt}]})

payload = json.dumps({
    "contents": contents,
    "generationConfig": {
        "maxOutputTokens": MAX_TOKENS,
        "temperature": 0.3,
    },
}).encode()

req = urllib.request.Request(
    url,
    data=payload,
    headers={"Content-Type": "application/json"},
    method="POST",
)

try:
    with urllib.request.urlopen(req, timeout=60) as resp:
        for raw_line in resp:
            line = raw_line.decode("utf-8").strip()
            if not line.startswith("data:"):
                continue
            data = line[5:].strip()
            if data == "[DONE]":
                break
            try:
                obj = json.loads(data)
                candidates = obj.get("candidates", [])
                for candidate in candidates:
                    content = candidate.get("content", {})
                    parts   = content.get("parts", [])
                    for part in parts:
                        text = part.get("text", "")
                        if text:
                            yield text
            except json.JSONDecodeError:
                continue

except urllib.error.HTTPError as e:
    body = e.read().decode()
    if e.code == 429:
        yield "⏳ Quota reached — using offline analysis:\n\n"
        yield from _offline_fallback("")
    else:
        yield "\n\nGemini API error " + str(e.code)
except Exception as e:
    yield "\n\nConnection error: " + str(e)
```

```
# ── Claude streaming ──────────────────────────────────────────────
def _stream_claude(
    system:  str,
    prompt:  str,
    history: list[dict] | None = None,
) -> Iterator[str]:
    key = _claude_key()
    messages = list(history or [])
    messages.append({"role": "user", "content": prompt})

    payload = json.dumps({
        "model":     CLAUDE_MODEL,
        "max_tokens": MAX_TOKENS,
        "system":    system,
        "messages":  messages,
        "stream":    True,
    }).encode()

    req = urllib.request.Request(
        CLAUDE_API_URL,
        data=payload,
        headers={
            "Content-Type":      "application/json",
            "x-api-key":         key,
            "anthropic-version": "2023-06-01",
        },
        method="POST",
    )

    try:
        with urllib.request.urlopen(req, timeout=60) as resp:
            for raw_line in resp:
                line = raw_line.decode("utf-8").strip()
                if not line.startswith("data:"):
                    continue
                data = line[5:].strip()
                if data == "[DONE]":
                    break
                try:
                    obj = json.loads(data)
                    if obj.get("type") == "content_block_delta":
                        delta = obj.get("delta", {})
                        if delta.get("type") == "text_delta":
                            yield delta.get("text", "")
                except json.JSONDecodeError:
                    continue
```

```
    except urllib.error.HTTPError as e:
        body = e.read().decode()
        yield f"\n\n Claude API error {e.code}: {body}"
    except Exception as e:
        yield f"\n\n Connection error: {e}"


# ── Offline fallback ──────────────────────────────────────────────────────
def _offline_fallback(prompt: str) -> Iterator[str]:
    lines = prompt.splitlines()

    def _extract(kw):
        for l in lines:
            if kw.lower() in l.lower():
                for p in l.split():
                    try: return float(p.replace("%","").replace(",",""))
                    except: pass
        return None

    n_anom  = _extract("Anomalies found")
    n_total = _extract("out of")
    rate    = (n_anom / max(n_total or 1, 1) * 100) if n_anom else None

    risk = "Low"
    if rate and rate > 8:   risk = "High"
    elif rate and rate > 4: risk = "Medium"

    feat_lines = [l.strip() for l in lines if l.strip().startswith(("1.","2.","3."))]
    top_feats  = ", ".join(feat_lines[:3]) if feat_lines else "the reported features"

    msg = f"""**Risk Assessment: {risk}**

{"%.1f%%" % rate if rate else "An elevated"} anomaly rate detected.

Key Factors
{top_feats}
Reasoning
The model flags observations statistically isolated from the majority distribution.
High-scoring samples deviate across multiple features simultaneously.

Recommendation
{"Investigate immediately — escalate to risk team." if risk in ("High","Critical") else "Monitor
flagged samples and review feature distributions."}

Offline mode. Set GEMINI_API_KEY or ANTHROPIC_API_KEY for AI responses."""

    for word in msg.split(" "):
        yield word + " "
```

Лістинг А.5 – Модуль пояснення точок даних (ai/point_explainer.py)

```
"""
ai/point_explainer.py
Builds structured context for a single clicked data point and sends it
to the LLM for instant explanation.

This is separate from the batch-level build_context() in llm_client.py.
It is optimised for:
  - low latency (shorter prompt, focused question)
  - per-point feature deviation analysis (z-score vs dataset mean)
  - structured output that copilot_panel can render with risk badges
"""
from __future__ import annotations
import numpy as np
from typing import Iterator
from ai.llm_client import stream_response, active_provider


# ── Point context builder ──────────────────────────────────────────────

def build_point_context(
    point_idx:     int,
    anomaly_score:  float,
    is_anomaly:    bool,
    features:      dict[str, float],
    feature_deltas: list[tuple[str, float]],
    feature_importance: list[dict],
    scenario:      str,
    model_name:    str,
    user_question:  str = "",
    chart_id:      str = "",
    **kwargs,
) -> tuple[str, str]:
    """
    Build (system_prompt, user_prompt) for a clicked chart point.

    feature_deltas: per-feature deviation from dataset mean in z-score units.
                    Positive = above mean. Sorted by |delta| descending.
    """
    label_str = "ANOMALY" if is_anomaly else "NORMAL"

    # Format per-feature values + deviations
    feat_lines = []
```

```
    for feat, delta in feature_deltas[:8]:
        raw_val  = features.get(feat, "?")
        sign     = "+" if delta >= 0 else ""
        raw_str  = f"{raw_val:.3f}" if isinstance(raw_val, float) else str(raw_val)
        feat_lines.append(
            f"  {feat}: {raw_str}  (Δ={sign}{delta:.2f}σ)"
        )

    # Global importance context
    top_global = []
    for r in feature_importance[:4]:
        fname = r.get("Feature") or r.get("feature") or "?"
        fimp  = float(r.get("Importance") or r.get("importance") or 0)
        top_global.append(f"  {fname} (global importance {fimp:.3f})")

    system = (
        "You are an expert anomaly detection analyst. "
        "A user has clicked a specific data point in a chart. "
        "Explain this individual point concisely and precisely. "
        "Format your response exactly as follows:\n"
        "RISK LEVEL: <CRITICAL|HIGH|MEDIUM|LOW>\n\n"
        "Key Factors:\n"
        "• <feature>: <value> (<deviation>% above/below normal)\n"
        "• ...\n\n"
        "Explanation: <2-3 sentences explaining why this point is anomalous or normal>\n\n"
        "Recommended Action: <1 concrete action>"
        "\n\nBe concise. Max 120 words total."
    )

    prompt = f"""POINT ANALYSIS REQUEST
Domain: {scenario}
Model: {model_name}
Classification: {label_str}
Anomaly score: {anomaly_score:.5f}
Point index: {point_idx}

Feature values and deviations from dataset mean:
{chr(10).join(feat_lines)}

Global feature importance (for reference):
{chr(10).join(top_global)}

{f"User question: {user_question}" if user_question else "Explain this specific data point."}
```

```
"""
    return system, prompt


# ── Streaming wrapper ──────────────────────────────────────────────────────

def stream_point_explanation(
    point_idx:     int,
    anomaly_score:  float,
    is_anomaly:    bool,
    features:      dict[str, float],
    feature_deltas: list[tuple[str, float]],
    feature_importance: list[dict],
    scenario:      str,
    model_name:    str,
    user_question:  str = "",
    chart_id:      str = "",   # passed from dashboard, not used in prompt
    **kwargs,                  # absorb any future extra keys
) -> Iterator[str]:
    """
    Stream an explanation for a clicked chart point.
    Falls back to offline explanation if no API key is set.
    """
    system, prompt = build_point_context(
        point_idx, anomaly_score, is_anomaly,
        features, feature_deltas, feature_importance,
        scenario, model_name, user_question,
    )

    if active_provider() == "offline":
        yield from _offline_point_fallback(
            point_idx, anomaly_score, is_anomaly, feature_deltas
        )
        return

    yield from stream_response(system, prompt, history=None)


# ── Offline fallback for point clicks ──────────────────────────────────────

def _offline_point_fallback(
    point_idx:    int,
    score:        float,
```

```
    is_anomaly:    bool,
    feature_deltas: list[tuple[str, float]],
) -> Iterator[str]:
    """Rule-based instant explanation when API key is absent."""
    label  = "ANOMALY" if is_anomaly else "NORMAL"

    # Risk level: use the maximum z-score deviation as the primary signal,
    # because anomaly scores are detector-specific and may be small in magnitude.
    # Also guaranteed at least MEDIUM if is_anomaly=True.
    max_delta = max((abs(d) for _, d in feature_deltas), default=0)
    if max_delta > 4.0 or (is_anomaly and max_delta > 3.0):
        risk = "CRITICAL"
    elif max_delta > 2.5 or (is_anomaly and max_delta > 1.5):
        risk = "HIGH"
    elif max_delta > 1.5 or is_anomaly:
        risk = "MEDIUM"
    else:
        risk = "LOW"

    # Top deviating features
    top = sorted(feature_deltas[:5], key=lambda x: abs(x[1]), reverse=True)[:3]
    bullets = []
    for feat, delta in top:
        pct    = int(abs(delta) * 34)        # rough % approximation from z-score
        direct = "above" if delta > 0 else "below"
        bullets.append(f"• {feat}: {pct}% {direct} normal range (Δ={delta:+.2f}σ)")

    bullet_str  = "\n".join(bullets) if bullets else "• No significant deviations detected"
    action      = (
        "Escalate to risk team for immediate review."
        if risk in ("CRITICAL", "HIGH") else
        "Flag for routine monitoring and secondary review."
        if risk == "MEDIUM" else
        "No immediate action required. Continue monitoring."
    )

    msg = f"""RISK LEVEL: {risk}

Key Factors:
{bullet_str}

Explanation: Point #{point_idx} is classified as {label} with anomaly score {score:.4f}. {"The
combination of multiple features deviating significantly from their baseline distributions triggered
```

```
the detector." if is_anomaly else "All feature values remain within expected statistical bounds for this dataset."}

Recommended Action: {action}

---
*Offline explanation. Set ANTHROPIC_API_KEY for AI-powered analysis.*"""

    for word in msg.split(" "):
        yield word + " "
```