

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ АВТОМАТИЧНОГО
ВИЯВЛЕННЯ ПРОГРАМНИХ ПОМИЛОК НА ОСНОВІ
ФОРМАЛІЗОВАНОЇ ПРОЄКЦІЇ ТРАЄКТОРІЙ ВИКОНАННЯ
ТА ІНТЕГРАЦІЇ ЛОГІВ І МЕТРИК¹Вінницький національний технічний університет

Досліджено проблему автоматичного виявлення програмних помилок в умовах, коли формальний аналіз забезпечує інтерпретованість, але не завжди є достатньо повним і обчислювально прийнятним, тоді як log-only і metrics-only підходи фіксують поведінкові відхилення, проте не гарантують узгодженості з формально визначеними властивостями виконання. Уперше запропоновано інформаційну технологію автоматичного виявлення програмних помилок, у якій логи виконання інтерпретуються як формалізована проєкція траєкторій виконання програм, а методи машинного навчання використовуються для апроксимації властивостей виконання, заданих у межах інтегрованої схеми «статичні сигнали – модель виконання – логи/метрики – ознаковий простір – поведінкова модель – індукція рішень». Запропонована побудова поєднує інваріантні, логові, метричні та динамічні ознаки й переводить задачу з ізольованого аналізу журналів подій або часових рядів метрик у задачу узгодженого розпізнавання дефектних режимів виконання. Експериментальну верифікацію виконано на відкритому датасеті LO2, який поєднує журнали подій і метричні характеристики виконання в мікросервісному середовищі. Отримані результати показали, що інтегрована модель як реалізація запропонованої інформаційної технології забезпечує значення F1-міри 0,742, що перевищує відповідні показники лог-орієнтованої базової моделі 0,657 і метрики-орієнтованої базової моделі 0,593. Площа під ROC-кривою становить 0,892, що вказує на вищу глобальну роздільну здатність моделі щодо безпечних і дефектних режимів виконання. Коефіцієнт узгодженості Кохена дорівнює 0,661, що підтверджує кращу відповідність автоматизованих рішень еталонній розмітці. Практичний ефект виявляється у понад двократному зменшенні частки випадків, які потребують експертної інтерпретації, та в істотному скороченні часу аналізу одного випадку. Абляційний аналіз додатково показав, що найбільший внесок у якість виявлення забезпечують логові та інваріантні компоненти ознакового простору, тоді як метричні та динамічні ознаки виконують стабілізувальну й уточнювальну функцію.

Ключові слова: автоматичне тестування програмного забезпечення, інформаційна технологія, аналіз логів, якісні метрики, машинне навчання, виявлення аномалій, поведінкове моделювання, мікросервісна архітектура.

Вступ

Сучасні програмні системи, особливо мікросервісні, функціонують у середовищі високої структурної мінливості, де окремий інцидент майже ніколи не редукується до однієї локальної події. Він формується як наслідок взаємодії кількох сервісів, каскадних переходів стану, часових затримок і ресурсних зрушень, що відображаються одночасно в логах виконання та метриках. У такому середовищі log-only підходи добре фіксують локальну семантику подій, але слабше відтворюють режимні зміни виконання, тоді як metrics-oriented моделі краще виявляють деградацію поведінки в часі, але втрачають локальну інтерпретованість критичного переходу. Саме ця подвійність і визначає практичну складність автоматичного виявлення програмних помилок у реальних системах. Вибір мікросервісного середовища зумовлений тим, що саме в таких системах програмна помилка часто



проявляється не в межах одного компонента, а через взаємодію кількох сервісів, часові залежності та зміни ресурсних характеристик, що одночасно відображаються в логах і метриках. Тому найбільший практичний ефект запропонованої технології очікується для розподілених програмних систем з інтенсивною міжсервісною взаємодією, значною кількістю журналів подій і метричних потоків та дефектами, прояв яких має комбінований подієво-динамічний характер. Додатково ситуацію ускладнює те, що сучасні log anomaly detection моделі переважно оцінюються як інструменти поведінкового виявлення, а не як компоненти формально інтерпретованої схеми аналізу виконання [1]—[4].

Актуальність проблеми посилюється тим, що в літературі вже зафіксовано дві стійкі тенденції. По-перше, дослідження log anomaly detection дедалі більше зміщуються від правил і ручного парсингу до sequence-based та language-model-based підходів, які дають високі результати на публічних бенчмарках, але залишаються переважно одновимірними з погляду джерел спостереження [5], [6]. По-друге, в області multivariate time series anomaly detection сформовано окремий клас metrics-oriented моделей, які працюють з часовими рядами метрик як із самодостатнім об'єктом аналізу, що підвищує чутливість до деградаційних режимів, але відриває рішення від семантики конкретних подій виконання [3], [7]—[9]. Отже, у суміжних напрямках накопичено значний арсенал моделей, але сам розрив між формальною інтерпретованістю виконання та емпіричною чутливістю моделей до поведінкових проявів поки не усунуто.

Цей розрив має не лише прикладний, а й власне науковий характер. Формальний аналіз програмного забезпечення дозволяє працювати з інваріантами, умовами досяжності та структурою виконання, проте у складних системах він або не охоплює всього простору реальних проявів, або вимагає надмірних обчислювальних витрат. Натомість машинно-навчальні моделі, що працюють на логах або метриках, можуть фіксувати реальні відхилення, але часто діють як поведінкові класифікатори без строгого зв'язку з формально визначеними властивостями виконання. Саме тому актуальною науковою проблемою є побудова такого підходу до автоматичного виявлення програмних помилок, який поєднував би формальну інтерпретованість моделі виконання з емпіричною чутливістю до логів і метрик у межах єдиної процедури індукції рішень. Інакше кажучи, проблема полягає не просто у підвищенні точності виявлення, а в усуненні суперечності між формальною коректністю, поведінковою інформативністю й практичною ефективністю автоматизованого аналізу.

Огляд log-only підходів показує, що вони є методично сильними в межах своєї модальності, але недостатніми як основа для повноцінної інформаційної технології автоматичного виявлення програмних помилок. У праці [10] журнали подій моделюються як послідовності шаблонів, а LSTM використовується для відновлення очікуваних патернів виконання. Сильна сторона цього підходу полягає в тому, що він переводить лог у форму послідовнісної моделі й дозволяє виявляти відхилення без ручного правила для кожного сценарію. Проте DeepLog залишається прив'язаним до логової траєкторії як до єдиного носія інформації, а тому не охоплює деградаційні режими, які виявляються через метрики раніше, ніж через семантично виразний логовий патерн. У [11] зроблено крок уперед за рахунок одночасного врахування sequential і quantitative anomalies усередині логового простору. Однак навіть там “quantitative” стосується статистики логових подій, а не повноцінного метричного представлення системи, тому проблема інтеграції різнотипних спостережень лишається нерозв'язаною. У [3] автори справедливо вказують на нестабільність реальних логів і пропонують семантичне представлення подій із attention-based Bi-LSTM, тобто долають важливе обмеження close-world assumption. Але навіть цей сильніший log-only підхід усе ще працює в межах логової модальності і не дає механізму узгодження з формально заданими властивостями виконання. Нарешті, [4] демонструє важливий перехід до end-to-end навчання на сирому тексті логів і показує, що нейромережа може автономно відтворювати деякі функції парсингу. Проте і тут предмет аналізу лишається лише логова поверхня виконання. Отже, log-only напрям еволюціонує від шаблонного послідовнісного моделювання до end-to-end семантичних представлень, але не усуває принципове обмеження, яке полягає у відсутності інтеграції з метричною динамікою та формальним апаратом властивостей виконання.

Не менш показовим є огляд metrics-oriented підходів, які репрезентують другий baseline-напрямок майбутньої експериментальної частини. У [5] запропоновано OmniAnomaly, який моделює нормальну поведінку багатовимірних часових рядів через стохастичну рекурентну архітектуру й реконструктивний критерій. Сильна сторона цього підходу полягає в здатності вловлювати латентну динаміку метрик і виявляти аномалії навіть за відсутності явних міток. Проте сам об'єкт аналізу тут уже агрегований до часових рядів, а локальна семантика подієвого переходу фактично втрачається. У [6] акцент перенесено на швидкість, стабільність і ефективність adversarially trained autoencoders.

Це важливо для практичного використання, але метод, як і OmniAnomaly, залишається всередині метричного простору й не має способу співвіднести виявлене відхилення з конкретною структурою логового сценарію. Робота [7] ще виразніше демонструє сильні сторони metrics-oriented напряду, оскільки використовує attention-based кодування для виявлення аномалій і діагностики на багатовимірних часових рядах. Проте навіть тут аномалія інтерпретується як поведінкове відхилення в динаміці метрик, а не як емпіричний прояв траєкторії виконання, пов'язаної з формально визначеною дефектною областю. У [8] ще раз підтверджується, що metrics-only моделі можуть бути ефективними в складних CPS і моніторингових сценаріях, але сам підхід не відповідає на питання, як узгодити генеративно або реконструктивно виявлену аномалію з логікою переходів виконання. Відтак metrics-oriented роботи добре моделюють деградацію режиму, але не дають інтерпретованого переходу від спостережуваного відхилення до формально осмисленого рішення про наявність програмної помилки.

Проведений критичний аналіз двох груп аналогів дозволяє сформулювати головну наукову суперечність, яка досі не подолана в наявних роботах. Log-only моделі зберігають локальну семантику подій і краще відбивають структуру виконання, але не охоплюють багатовимірну поведінкову динаміку системи. Metrics-oriented моделі, навпаки, добре виявляють режимні зсуви в часових рядах, але втрачають семантику подієвого переходу й формальну прив'язку до властивостей виконання. Формальні підходи забезпечують інтерпретованість, але не дають достатньо гнучкого механізму для роботи з реальними емпіричними проявами, зокрема з шумними, неповними й динамічно змінними логами та метриками. Отже, для розв'язання актуальної наукової проблеми в статті:

- уперше запропоновано інформаційну технологію автоматичного виявлення програмних помилок, у якій логи та метрики інтерпретуються як узгоджена спостережувана проєкція траєкторій виконання, що, на відміну від log-only і metrics-oriented підходів, забезпечує спільне врахування семантики подій і поведінкової динаміки системи;
- удосконалено спосіб формування ознакового представлення виконання шляхом інтеграції інваріантних, логових, метричних і динамічних характеристик у єдиному просторі, що, на відміну від найближчих аналогів, забезпечує зв'язок емпіричних проявів дефекту з формально визначеними властивостями виконання;
- розвинуто підхід до поведінкового виявлення дефектних режимів, у якому машинне навчання використовується не як автономний засіб виявлення аномалій, а як механізм апроксимації формально визначених властивостей виконання в межах послідовності «траєкторія – проєкція – сегмент – ознаковий простір – рішення».

Постановка дослідження

Об'єктом дослідження є процес автоматичного виявлення програмних помилок у програмному забезпеченні під час його виконання в складних, зокрема мікросервісних, середовищах.

Предметом дослідження є інформаційна технологія автоматичного виявлення програмних помилок, а також моделі та методи формалізації траєкторій виконання програм, побудови інтегрованого ознакового простору на основі логів і метрик та прийняття рішень щодо належності виконання до дефектних режимів.

Методи дослідження охоплюють методи формалізації процесів виконання програм у вигляді траєкторій, методи побудови інтегрованого ознакового простору на основі логів і метрик, методи машинного навчання для моделювання поведінки системи, а також методи статистичного оцінювання якості класифікації, узгодженості рішень і ефективності автоматизації аналізу.

Метою дослідження є підвищення ефективності автоматичного виявлення програмних помилок шляхом розроблення інформаційної технології, що забезпечує підвищення якості класифікації, підвищення узгодженості автоматизованих рішень та зменшення частки випадків, які потребують експертної інтерпретації, і скорочення часових витрат на аналіз.

Для досягнення поставленої мети необхідно виконати такі завдання:

- формалізувати процес виконання програмного забезпечення у вигляді траєкторій виконання та визначити спосіб інтерпретації логів і метрик як їх спостережуваної проєкції;
- розробити інтегрований ознаковий простір, що поєднує інваріантні, логові, метричні та динамічні характеристики виконання;
- побудувати поведінкову модель, яка апроксимує формально визначені властивості виконання;
- розробити процедуру прийняття рішення щодо належності траєкторій виконання до

дефектних режимів;

– сформувані експериментальний конвеєр верифікації інформаційної технології на основі відкритого датасету;

– здійснити експериментальне оцінювання ефективності запропонованої технології за показниками якості класифікації, узгодженості рішень і практичної ефективності;

– виконати порівняльний аналіз із базовими підходами та визначити внесок окремих компонентів інтегрованої моделі в загальний результат виявлення програмних помилок.

Моделі й методи

Розглянемо автоматичне виявлення програмних помилок як задачу встановлення досяжності дефектних режимів виконання та їх розпізнавання за експлуатаційними даними. Така постановка виходить з того, що програмна помилка не повинна ототожнюватися ні з локальним дефектом програмного коду, ні з окремим записом журналу подій. Дефект задає лише потенційну можливість порушення коректності. Помилка виникає на конкретній траєкторії виконання; лог і метрика фіксують лише спостережувані наслідки цього процесу. Відтак центральним об'єктом аналізу є не окрема подія журналювання, а траєкторія виконання програми, для якої можна встановити, чи належить вона до безпечної чи до дефектної області. Логи й метрики в цій побудові виконують роль формалізованої проєкції виконання, а методи машинного навчання використовуються для апроксимації формально визначених властивостей виконання в тих зонах простору трас, де прямий формальний аналіз є неповним або обчислювально надмірним.

Нехай програму подано системою переходів станів

$$P = \langle S, s_0, F, T \rangle, \quad (1)$$

де S є множиною досяжних станів виконання, $s_0 \in S$ — початковим станом, $F : S \rightarrow 2^S$ — оператором переходів, $T \subseteq S$ — множиною завершальних станів. В контексті моделі (1) стан $s \in S$ розуміється як повний щодо вибраного рівня абстракції опис поточної конфігурації програми: значень змінних, станів пам'яті, контексту керування, параметрів взаємодії з ресурсами та зовнішніми сервісами, а також інших характеристик, від яких залежить коректність виконання. Таке подання є необхідним, оскільки дозволяє описувати помилку не як текстову або синтаксичну аномалію, а як порушення властивостей процесу виконання.

Множину всіх допустимих трас виконання визначимо як

$$\Gamma(P) = \{ \tau = \langle s_0, s_1, \dots, s_n \rangle : s_{k+1} \in F(s_k), k = 0, \dots, n-1 \}. \quad (2)$$

Кожна траєкторія $\tau \in \Gamma(P)$ є скінченною послідовністю станів, упорядкованих відповідно до фактичної або потенційної динаміки виконання, а P позначає всю програмну систему, подану через (1). Саме множина $\Gamma(P)$, а не множина окремих станів, є базовим простором для постановки задачі виявлення помилок. Це пов'язано з тим, що значна частина дефектів не проявляється в окремому стані ізольовано, а виникає як наслідок певної послідовності переходів, наприклад, повторного доступу до звільненого ресурсу, поступової деградації часових характеристик або накопичення некоректних проміжних значень.

Для зв'язку між формальною моделлю виконання і доступними під час експлуатації даними вводиться простір спостережень $R = R_{\log} \times R_{\text{met}}$, де $R_{\log}$ є простором лог-подій, а R_{met} — простором метричних характеристик виконання. Оператор спостереження

$$\Pi : S \rightarrow R \quad (3)$$

ставить кожному стану $s \in S$ у відповідність пару $\Pi(s) = (\ell(s), m(s))$, де $\ell(s)$ репрезентує подієву компоненту, а $m(s)$ — метричну компоненту. На рівні траєкторій оператор Π індукує відображення

$$L(\tau) = (\Pi(s_0), \Pi(s_1), \dots, \Pi(s_n)), \quad (4)$$

яке задає лог-метричну трасу, що спостерігається під час виконання. Формула (4) фіксує головний принцип подальшої побудови: логи і метрики є не самостійним об'єктом класифікації, а проєкцією траєкторії виконання у простір спостережень. Саме тому інтерпретація експлуатаційних даних не може бути коректною без прив'язки до простору трас. Той самий запис типу ERROR, timeout або HTTP 500 може виникати в різних контекстах і відповідати як реальному дефектному режиму, так

і зовнішньому збуренню, яке не порушує внутрішніх інваріантів програми.

У математичному сенсі це означає, що оператор Π загалом не є ін'єктивним: різні стани можуть мати однакову лог-метричну репрезентацію. Відтак задача виявлення помилки не може бути зведена до прямої класифікації елементів простору R , вона потребує залучення інформації про структуру траєкторії, яка породила відповідну послідовність спостережень. Саме цей факт зумовлює необхідність інтеграції формального аналізу з поведінковим.

Нехай $\Phi = \{\varphi_1, \dots, \varphi_q\}$ — множина інваріантів коректності, де кожний інваріант $\varphi_i : S \rightarrow \{0, 1\}$ визначає локальну умову безпечного виконання. Тоді множина коректних трас задається співвідношенням

$$\Gamma_{safe} = \{\tau \in \Gamma(P) : \forall s_k \in \tau \forall i \varphi_i(s_k) = 1\}, \quad (5)$$

а множина дефектних трас визначається як

$$\Gamma_{err} = \Gamma(P) \setminus \Gamma_{safe}. \quad (6)$$

Формули (5)—(6) задають строгий зміст поняття програмної помилки: помилковою є не подія в журналі й не окремий фрагмент коду, а траєкторія, на якій порушується хоча б один із інваріантів коректності. Таке подання є принципово важливим для всієї технології, оскільки дозволяє розділити три різні рівні аналізу. По-перше, дефект програмного коду може існувати як потенційна причина, але не реалізуватися на конкретному запуску. По-друге, помилка виникає тоді, коли відповідний дефект досягається на певній траєкторії виконання. По-третє, лише після цього вона отримує спостережуване відображення в логах і метриках. Саме це розмежування унеможливує редукцію задачі до поверхневого розпізнавання патернів у журналі подій.

Для кожного попередження статичного аналізу $w \in W$ вводиться множина релевантних трас $\Gamma(w) \subseteq \Gamma(P)$, які проходять через програмні області, пов'язані з відповідним сигналом. Якщо є предикатом шляху, що виражає здійсненність траєкторії τ , то умова досяжності програмної помилки відносно сигналу w має вигляд

$$\exists \tau \in \Gamma(w) : PC(\tau) \wedge (\tau \in \Gamma_{err}). \quad (7)$$

Формула (7) визначає формальний рівень інформаційної технології. Вона означає, що попередження статичного аналізу вважається підтвердженням не за фактом своєї появи, а лише за наявності здійсненої дефектної траєкторії, на якій порушується інваріант коректності. Така постановка дає можливість відокремити хибнопозитивні попередження від тих, що справді відповідають досяжним помилкам виконання.

Попри строгість умови (7), у реальних програмних системах повна перевірка досяжності є або надто дорогою, або не вичерпує всіх поведінкових проявів дефекту. Багато помилок мають не лише логічну, а й часову, ресурсну або статистичну складову. Саме тому наступний рівень технології працює не з усією трасою $L(\tau)$ як неподільним об'єктом, а з її сегментами.

Нехай $\Sigma_h : L(\tau) \mapsto (\sigma_1, \dots, \sigma_r)$ є оператором сегментації, який ділить лог-метричну трасу на фрагменти довжини h . Тут σ_j відповідає локальному вікну спостережень, в контексті якого можна аналізувати мікродинаміку виконання: зміну метрик, характер переходів між лог-подіями, повторюваність викликів, накопичення затримок або появу критичних сигналів. Введення сегментації має принциповий сенс. Практичне автоматичне виявлення помилок здійснюється не щодо всієї траєкторії одразу, а щодо локальних спостережуваних фрагментів, які несуть інформацію про перехід у дефектний режим. Наприклад, якщо траєкторія транзакційного сервісу містить довгу послідовність коректних переходів, а дефект проявляється лише на обмеженому часовому інтервалі, то саме локальний сегмент містить релевантний сигнал для виявлення. Без сегментації цей сигнал може бути «розмитий» усередненням по всій трасі й втратити інформативність.

Для перенесення сегментів трас у простір обчислюваних представлень вводиться ознакове відображення

$$\Psi : \Sigma_h(L(\tau)) \rightarrow \mathcal{X} \subseteq \mathbb{R}^d, \quad (8)$$

де $\mathcal{X}$ є ознаковим простором. Важливо, що областю визначення Ψ є не окремий лог-запис і не вся траєкторія без структури, а саме сегмент лог-метричної траси. Це забезпечує принципову прив'язку ознак до локальних режимів виконання.

Оператор Ψ задається як композиція чотирьох змістовно різних блоків

$$\Psi = \Psi_{inv} \oplus \Psi_{log} \oplus \Psi_{met} \oplus \Psi_{dyn}, \quad (9)$$

де Ψ_{inv} формує інваріантні ознаки, Ψ_{log} — подієві ознаки, Ψ_{met} — метричні ознаки, а Ψ_{dyn} — динамічні ознаки.

Інваріантний блок Ψ_{inv} формувався на рівні сегмента шляхом перевірки виконання формально заданих інваріантів $\phi_i \in \Phi$ для станів, представлених у відповідному сегменті спостережень. Зокрема, ознака `inv_violation_flag` набувала значення 1, якщо в межах сегмента фіксувалося порушення хоча б одного інваріанта ϕ_i , і 0 — якщо виконувалися всі інваріанти, що перевіряються. Ознака `error_status_consistency` характеризувала узгодженість між наявністю формально виявленого порушення інваріанта та спостережуваними у сегменті ознаками помилкового стану за логовими подіями і статусами виконання. Отже, інваріантні ознаки формувалися детерміновано з доступних спостережень і не використовували тестові мітки класу.

Інваріантний блок Ψ_{inv} переносить у простір ознак інформацію про порушення або наближення до порушення інваріантів коректності. Ці ознаки забезпечують безпосередній зв'язок між формальною моделлю помилки, заданою формулами (5)—(6), і поведінковим рівнем. Подієвий блок Ψ_{log} відображає структуру лог-подій: частоти появи певних класів повідомлень, наявність критичних ланцюжків подій, повторюваність викликів і типові логічні патерни переходів. Метричний блок Ψ_{met} кодує кількісні характеристики сегмента: середні значення, піки, дисперсії, прирости та співвідношення між параметрами продуктивності, використання пам'яті, інтенсивності звернень тощо. Нарешті, динамічний блок Ψ_{dyn} фіксує темпоральну організацію спостережень: часові інтервали між подіями, швидкість зміни метрик, ритм повторень і ознаки деградації режиму в часі. Отже, вектор $x = \Psi(\sigma)$ не є просто технічним кодуванням журналу, а є структурованим образом локального фрагмента виконання. У цьому полягає головний зміст формули (9): оператор Ψ не лише зменшує розмірність спостережуваних даних, а й переносить формально визначені властивості виконання в простір ознак, придатний для автоматизованого аналізу.

На рівні множин формула (8) індукує образи $\mathcal{X}_{safe} = \Psi(\Sigma_h(L(\Gamma_{safe})))$, $\mathcal{X}_{err} = \Psi(\Sigma_h(L(\Gamma_{err})))$. Саме ці області є геометричними образами безпечних і дефектних режимів виконання в ознаковому просторі. Якщо побудова Ψ є адекватною, то $\mathcal{X}_{safe}$ та $\mathcal{X}_{err}$ мають бути достатньо розрізненими для подальшої апроксимації.

Нехай χ_{err} є характеристичною функцією дефектної області в просторі трас, тобто $\chi_{err}(\tau) = 1$ для $\tau \in \Gamma_{err}$ і $\chi_{err}(\tau) = 0$ для $\tau \notin \Gamma_{err}$. Тоді машинно-навчальна модель визначається як відображення

$$M: \mathcal{X} \rightarrow [0,1], \quad (10)$$

яке апроксимує належність сегмента до образу дефектної області $\mathcal{X}_{err}$. У цьому поданні $M(x)$ є оцінкою того, наскільки спостережуваний сегмент виконання узгоджується з дефектним режимом. Це означає, що машинне навчання не виступає автономним класифікатором логів. Воно реалізує апроксимаційний механізм, який продовжує формальний аналіз із простору трас у простір спостережуваних представлень. Іншими словами, M не створює поняття помилки заново, а відновлює вже визначену формально дефектну область у тій геометрії, яку задає оператор Ψ .

Нехай $\Gamma_{train} \subseteq \Gamma(P)$ — множина трас, для яких належність до Γ_{err} або Γ_{safe} встановлено формальним шляхом. Тоді навчання моделі формулюється як задача мінімізації регуляризованого емпіричного ризику

$$\mathcal{J}(M) = \frac{1}{|\Gamma_{train}|} \sum_{\tau \in \Gamma_{train}} \sum_{\sigma \in \Sigma_h(L(\tau))} \mathcal{L}(M(\Psi(\sigma)), y(\sigma)) + \lambda \Omega(M), \text{ де } y(\sigma) \in \{0,1\} \text{ — мітка сегмента, індукована формальною належністю відповідної траєкторії до безпечної або дефектної області; } \mathcal{L} \text{ — функція втрат, } \Omega(M) \text{ — регуляризатор складності моделі; } \lambda \geq 0 \text{ — коефіцієнт регуляризації. Така постановка є принциповою, оскільки показує: навчальна вибірка формується не довільною ручною розміткою логів, а формально верифікованими трасами й їх сегментами.}$$

Наведена побудова потребує двох базових умов коректності.

Твердження 1. Якщо для будь-яких $\tau_1 \in \Gamma_{safe}$, $\tau_2 \in \Gamma_{err}$ та відповідних сегментів $\sigma_1 \in \Sigma_h(L(\tau_1))$,

$\sigma_2 \in \Sigma_h(L(\tau_2))$ виконується $\Psi(\sigma_1) \neq \Psi(\sigma_2)$, то області $\mathcal{X}_{safe}$ і $\mathcal{X}_{err}$ є розрізненими, а задача апроксимації дефектної області в просторі $\mathcal{X}$ є коректно поставленою.

Зміст цього твердження полягає в тому, що оператор Ψ повинен зберігати структурну роздільність між безпечними й дефектними режимами. Якщо це не виконується, жодна модель, що працює лише на рівні $\mathcal{X}$, не може забезпечити коректне розділення класів.

Твердження 2. Якщо для деякого попередження $w \in W$ виконується $\Gamma(w) \cap \Gamma_{err} = \emptyset$, то будь-яка додатна реакція моделі M на сегментах трас з $\Gamma(w)$ не може інтерпретуватися як формально підтверджена програмна помилка й повинна розглядатися як поведінкова аномалія або як наслідок недостатньої роздільної здатності ознакового відображення.

Це твердження фіксує підпорядкований статус машинно-навчального рівня щодо формального. Отже, позитивне рішення моделі набуває змісту підтвердженої програмної помилки лише тоді, коли воно узгоджується з умовою досяжності, заданою формулою (7).

З урахуванням побудованих об'єктів інтегровану інформаційну технологію автоматичного виявлення програмних помилок доцільно визначити як відображення

$$\mathcal{T} : W \times \Gamma(P) \rightarrow \{0, 1\}, \quad (11)$$

$$\text{де} \quad \mathcal{T}(w, \tau) = \begin{cases} 1, \forall \tau \in \Gamma(w), \tau \in \Gamma_{err}, \exists \sigma \in \Sigma_h(L(\tau)) : M(\Psi(\sigma)) \geq \theta, \\ 0, \text{otherwise,} \end{cases} \quad (12)$$

$\theta \in [0, 1]$ є пороговим значенням, яке відокремлює дефектну область у просторі ознак від безпечної. Формули (11)—(12) виражають головний зміст усієї технології. Рішення щодо наявності помилки приймається лише за одночасного виконання трьох умов. По-перше, траєкторія повинна бути релевантно певному статичному сигналу w . По-друге, ця траєкторія повинна належати до дефектної області у формальному сенсі. По-третє, хоча б один її спостережуваний сегмент повинен потрапляти до дефектної області в ознаковому просторі з погляду моделі M . Саме така композиція й реалізує схему «статичні сигнали – модель виконання – логи/метрики – ознаковий простір – поведінкова модель – індукція рішень».

Для кількісної оцінки якості побудованої технології розглядається валідаційна вибірка $\mathcal{D}_{val} = (\tau_j, w_j, y_j)_{j=1}^N$, де $y_j \in \{0, 1\}$ є еталонною міткою, що відображає формально або експертно підтверджену належність траєкторії до дефектної області. Для кожного елемента вибірки технологія породжує передбачення $\hat{y}_j = \mathcal{T}(w_j, \tau_j)$. На цій основі визначаються істинно-додатні, хибнододатні, істинно-від'ємні та хибновід'ємні випадки, що дозволяє ввести стандартні показники класифікаційної якості.

Точність визначається за формулою

$$P = \frac{TP}{TP + FP}, \quad (13)$$

а повнота — формулою

$$R = \frac{TP}{TP + FN}. \quad (14)$$

Тут істинно-додатні випадки TP відповідають траєкторіям, що належать Γ_{err} і правильно ідентифіковані технологією. Хибнододатні FP відображають випадки, коли рішення $\mathcal{T}$ сигналізує про помилку за її відсутності. Хибновід'ємні FN відповідають пропущеним дефектним режимам виконання.

Показник $F1$ -міри задається як гармонійне середнє точності й повноти

$$F1 = \frac{2 \cdot P \cdot R}{P + R}. \quad (15)$$

Формули (13)—(15) є природними саме для побудованої технології: вони оцінюють, наскільки збалансованими є її рішення з погляду, з одного боку, пропуску реальних помилок, а з іншого — хибних спрацьовувань.

Оскільки рішення моделі M залежать від порога θ , індукований нею оператор $\mathcal{T}$ задає сім'ю

класифікаторів. Це дозволяє ввести істинно-додатну й хибнододатну частки: $TPR(\theta) = \frac{TP(\theta)}{TP(\theta) + FN(\theta)}$;

$FPR(\theta) = \frac{FP(\theta)}{FP(\theta) + TN(\theta)}$, та визначити ROC-характеристику як множину точок

$$ROC = \{(FPR(\theta), TPR(\theta)) : \theta \in [0, 1]\}. \quad (16)$$

Відповідно, площа під ROC-кривою відображає роздільну здатність ознакового простору $\mathcal{X}$ і моделі M щодо відокремлення дефектних режимів від безпечних.

Для оцінювання узгодженості автоматизованих рішень із зовнішнім еталоном використовується коефіцієнт Кохена $\kappa = \frac{p_o - p_e}{1 - p_e}$, де p_o є фактичною часткою збігів між автоматизованими й етало-

ними рішеннями, а p_e — очікуваною часткою збігів за випадкового узгодження. Показник κ важливий у цій роботі, оскільки він дозволяє оцінити не лише класифікаційну якість, а й ступінь узгодженості індукованих рішень із формальною або експертною інтерпретацією.

Оскільки інформаційна технологія спрямована не тільки на коректне виявлення помилок, а й на зменшення навантаження на експерта, вводиться коефіцієнт скорочення експертної інтерпретації

$$\rho_E = \frac{E_{base}}{E_T}, \quad (18)$$

де E_{base} — кількість випадків, що потребують експертного аналізу без застосування технології, а E_T — кількість таких випадків після її застосування. Аналогічно коефіцієнт часової ефективності

$$\rho_T = \frac{T_{base}}{T_T} \quad (19)$$

порівнює середній час аналізу одного випадку без технології та з її застосуванням. Формули (18)—(19) безпосередньо переводять теоретичну коректність побудованого апарату в площину практичної ефективності.

Розглянемо сервіс обробки транзакцій, для якого статичний аналіз сформував попередження w_j про можливий вихід за межі буфера в модулі серіалізації. Це попередження індукує множину релевантних трас $\Gamma(w_j)$. Нехай існує траєкторія $\tau \in \Gamma(w_j)$, для якої предикат шляху є здійсненим і на якій порушується інваріант адресної допустимості, тобто $\tau \in \Gamma_{err}$. Отже, на формальному рівні попередження має досягнути дефектну реалізацію.

Траєкторія τ відображається оператором L у лог-метричну трасу $L(\tau)$. У цій трасі фіксуються звернення до буфера, аномальне зростання затримки обробки, серія повторних викликів, зміна контрольної довжини й подальший аварійний сигнал. Після застосування оператора сегментації один із сегментів $\sigma \in \Sigma_h(L(\tau))$ виявляється інформативним для аналізу. Оператор Ψ відображає його у вектор $x = \Psi(\sigma)$, де інваріантні, подієві, метричні й динамічні компоненти узгоджено вказують на дефектний режим. Якщо $M(x) \geq \theta$, то з формули (12) випливає $T(w_j, \tau) = 1$. У цьому прикладі програмна помилка встановлюється не як ізольований лог-патерн, а як результат проходження через усі рівні інформаційної технології — від статичного сигналу й формальної досяжності до поведінкового підтвердження на сегменті лог-метричної траси.

Запропонований математичний апарат визначає інформаційну технологію автоматичного виявлення програмних помилок як послідовність формально пов'язаних перетворень: від простору виконань і множини траєкторій до простору спостережень, далі до структурованих ознакових представлень і, нарешті, до індукції рішення про наявність дефектного режиму. Формальна дефектна область Γ_{err} задає строгий зміст поняття помилки, лог-метрична траса $L(\tau)$ забезпечує її спостережувану репрезентацію, оператор Ψ переносить цю репрезентацію у векторний простір, модель M апроксимує дефектну область на рівні ознак, а відображення T , задане формулами (11)—(12), завершує побудову рішення. У такому вигляді секція не лише задає формальний базис технології, а й безпосередньо готує перехід до експериментальної валідації за показниками (15)—(19).

Експерименти та результати

Емпірична верифікація запропонованої інформаційної технології має перевірити не загальну здатність моделі розпізнавати аномалії в журналах подій, а значно конкретнішу властивість. Ідеться про те, чи забезпечує композиція, формалізована у (1)—(14), стійке покращення виявлення дефектних режимів виконання тоді, коли логи та метрики розглядаються як узгоджена проєкція траєкторій виконання, а не як дві незалежні модальності спостереження. Саме тому експериментальна частина будуватиметься не навколо одного класифікатора, а навколо повної процедури

$$\tau \mapsto L(\tau) \mapsto \Sigma_h(L(\tau)) \mapsto \Psi(\sigma) \mapsto M(\Psi(\sigma)) \mapsto T(w, \tau),$$

після чого рішення оцінюються метриками (15)—(19).

Вибір датасету підпорядкований саме цій логіці. Відкрита вибірка LO2 [https://zenodo.org/records/14938118] є придатною не тому, що містить великий обсяг журналів подій як таких, а тому, що вона поєднує логи сервісів, метричні характеристики виконання та мітки помилкових сценаріїв у межах однієї мікросервісної системи. Це дозволяє інтерпретувати окремий запуск як спостережувану реалізацію $\tau \in \Gamma(P)$, а лог-метричний запис цього запуску — як емпіричну реалізацію $L(\tau)$. Відповідно, LO2 підтримує саме ту форму перевірки, яка потрібна для моделі секції 3. У відкритих наборах p чистимилогами, зокрема в HDFS із LogHub [https://www.kaggle.com/datasets/ayenuryrr/loghub-hdfs-hadoop-distributed-file-system-data], можна перевіряти лише частину конструкції, пов'язану з $\Psi_{\log}$, тоді як перевірка інваріантно-логіко-метрично-динамічної композиції в повному вигляді там неможлива. У цьому сенсі вибір LO2 є не зручним, а методично необхідним.

Перш ніж аналізувати числові результати, доцільно зафіксувати загальну архітектуру експериментальної перевірки, оскільки саме вона пов'язує теоретичний апарат (1)—(14) з фактичними обчисленнями метрик (15)—(19). На цьому етапі важливо не лише показати етапи конвеєра, а й уточнити, що кожен з них є операціоналізацією відповідного блоку математичної моделі. Зокрема, LO2 задає спостережувані реалізації $L(\tau)$, сегментація реалізує Σ_h , простір ознак інстанціює Ψ , поведінкова модель відповідає оператору M , а модуль прийняття рішення реалізує T . Саме тому рис. 1 є методичним центром цієї секції.

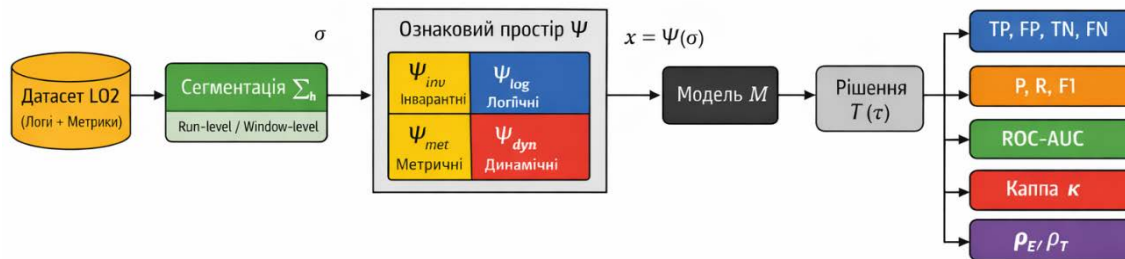


Рис. 1. Фреймворк для верифікації інформаційної технології

Рис. 1 показує, що верифікація побудована як ланцюг послідовних перетворень, у якому інформація втрачає необроблений емпіричний характер і послідовно набуває формалізованого змісту. На лівому фланзі схеми LO2 подає спостереження, які ще не є рішеннями. Після сегментації Σ_h формується локальний рівень аналізу, придатний для відображення в $\mathcal{X}$. Далі блоковий оператор Ψ інтегрує інваріантні, логічні, метричні та динамічні ознаки, а модель M переводить це представлення у шкалу ризику дефектного режиму. Важливим є те, що права частина конвеєра не обмежується $F1$ чи $ROC-AUC$. Тут явно присутні TP , FP , TN , FN , з яких вже виводяться P , R , $F1$, далі $ROC-AUC$, κ , ρ_E та ρ_T . Отже, рис. 1 фіксує повний шлях від спостереження до статистично оціненого рішення, а не просто послідовність етапів обробки даних.

Після цього методично доцільно розділити опис властивостей самих даних і властивостей протоколу верифікації. Таке розмежування принципове, оскільки інакше ефекти, зумовлені дисбалансом класів, розмірністю ознакового простору чи щільністю сегментації, можуть бути хибно приписані моделі. Спочатку фіксуються характеристики вибірки, оскільки саме вони визначають, наскільки складною є задача виявлення дефектних режимів.

Еталонні мітки класів формувалися на основі наявної в LO2 розмітки запусків: коректним запуском відповідала мітка $y = 0$, а запуском із інжектіваним/зафіксованим дефектним сценарієм —

$y = 1$. Після window-level сегментації сегменти успадковували мітку відповідного запуску. Розбиття на train, validation і test виконувалося зі збереженням установленної розмітки.

Табл. 1а показує, що робоча вибірка містить 200 запусків, з яких 126 є коректними, а 74 — помилковими. Частка помилкового класу дорівнює 37,0 %, отже задача від самого початку є дисбалансованою. Це одразу пояснює, чому для оцінювання не використовувалася асигасу, а центральними стали $F1$, $ROC-AUC$ і κ , які краще поведуться в умовах асиметричного розподілу.

Таблиця 1а

Характеристика датасету

Показник	Позначення	Значення	Примітка
Назва датасету	—	LO2 sample	відкрита вибірка
Джерело	—	Zenodo	публічний доступ
Кількість запусків	N_{runs}	200	після фільтрації
Кількість коректних запусків	N_{normal}	126	—
Кількість помилкових запусків	N_{faulty}	74	—
Частка помилкових	—	37,0 %	дисбаланс класів
Кількість сервісів	N_{services}	7	—
Кількість типів помилок	N_{tags}	53	—
Кількість метрик (загальна)	$N_{\text{metrics}}^{\text{raw}}$	485	—
Кількість метрик (відібраних)	N_{metrics}	24	після відбору
Частка пропусків метрик	—	4,2 % $\pm$ 1,1 %	—
Кількість лог-подій	N_{log}	$2,11 \cdot 10^6 \pm 1,4 \cdot 10^5$	—
Довжина сегмента (логів)	—	1684 ± 301	рядків
Медіана довжини сегмента	—	1651	—
Інтерквартильний розмах	—	1488—1879	—

Другий важливий результат табл. 1а полягає в масштабі логових даних. Загальний обсяг спостережень перевищує $2,11 \cdot 10^6$ подій, а середня довжина сегмента становить 1684 рядки за стандартного відхилення 301. Це означає, що Ψ_{log} і Ψ_{dyn} будуються не на обмежених шаблонних фрагментах, а на досить насичених локальних представленнях. Третій критичний момент пов'язаний з метричним простором. Перехід від 485 сирих метрик до 24 відібраних ознак показує, що метричний компонент не переобтяжує Ψ надмірною корельованою інформацією. Водночас наявність 4,2 % пропусків за стандартного відхилення 1,1 % свідчить, що метричний канал не є ідеальним і тому не може самостійно забезпечити стійку класифікацію.

Параметри самого протоколу потрібно розглядати окремо, оскільки вони визначають не складність даних, а жорсткість процедури перевірки. Для цього вводиться табл. 1б, яка фіксує ті компоненти експерименту, від яких безпосередньо залежить відтворюваність оцінок.

Таблиця 1б

Параметри протоколу експерименту

Параметр	Позначення	Значення	Примітка
Режим сегментації	—	window-level	основний
Довжина вікна	h	60 с	—
Кількість сегментів	N_{segments}	1348	після обробки
Сегментів на запуск	—	$6,74 \pm 2,10$	—
Медіана сегментів	—	7	—
Інтерквартильний розмах	—	5–8	—
Train / Validation / Test	—	60 / 20 / 20 %	стратифіковано
Кількість повторів	—	10	—
Вибір порогу	—	max F1	на validation
Поріг класифікації	θ	$\approx 0,52$ — $0,57$	залежно від моделі
Відбір ознак	—	hybrid	комбінований
Проекція ознак	—	PCA / UMAP	для візуалізації
Область навчання проєкції	—	train	без витоків даних
Політика випадковості	—	фіксовані seed	відтворюваність

Табл. 1b показує, що основний експеримент побудовано в режимі window-level segmentation з фіксованою довжиною вікна $h = 60$ с. Саме через цей вибір застосування Σ_h є змістовно узгодженим з секцією 2, де Ψ діє не на весь запуск як монолітний об'єкт, а на локальні сегменти виконання. Після сегментації та очищення формується 1348 сегментів, а середня кількість сегментів на запуск дорівнює 6,74 за стандартного відхилення 2,10. Отже, експеримент не є надто грубим, але й не зводиться до надмірно дрібної нарізки, яка руйнувала б структуру траєкторії. Друга характеристична особливість протоколу полягає у стратифікованому розбитті 60/20/20 та десятикратному повторенні. Це означає, що подальші результати основані не на одиничній конфігурації train/validation/test, а на статистично повторюваній процедурі. Третій важливий параметр пов'язаний із вибором порога θ . Його встановлено за правилом максимізації $F1$ на validation, тобто рішення $\mathcal{T}$ не оптимізуються за тестовою вибіркою і не містять витоку інформації.

Після фіксації вихідних даних і протоколу можна перейти до головного зіставлення, яке фактично визначає емпіричну цінність усієї інформаційної технології. Йдеться про порівняння трьох конфігурацій, а саме, аналог, орієнтований на аналіз логів, (log-only baseline), аналог, орієнтований на аналіз метрик, (metrics-only baseline) та інтегрована реалізація запропонованої інформаційної технології (integrated model), яка включає повну структуру $\Psi_{inv} \oplus \Psi_{log} \oplus \Psi_{met} \oplus \Psi_{dyn}$.

У проведених експериментах поведінкову модель M реалізовано на основі градієнтного бустингу над деревами рішень XGBoost з параметрами $n_{estimators} = 200$, $max_depth = 4$, $learning_rate = 0,05$, $subsample = 0,8$, $colsample_bytree = 0,8$ та цільовою функцією binary:logistic. Навчання виконувалося на train-частині кожного стратифікованого розбиття 60/20/20, вибір порога θ здійснювався на validation-частині за критерієм максимізації $F1$, а остаточні показники обчислювалися виключно на test-частині. Процедуру повторювали десять разів з фіксованими значеннями seed.

Таблиця 2

Основні результати верифікації інформаційної технології

Метрика	Log-only	Metrics-only	Integrated
Precision (P)	0,701	0,644	0,781
Recall (R)	0,621	0,556	0,709
$F1$ -score	0,657	0,593	0,742
$F1$ SD	0,051	0,059	0,046
95 % CI ($F1$)	[0,625–0,689]	[0,556–0,630]	[0,714–0,770]
ROC-AUC	0,824	0,781	0,892
ROC SD	0,037	0,046	0,031
95 % CI (ROC-AUC)	[0,801–0,847]	[0,752–0,810]	[0,873–0,911]
Cohen's κ	0,517	0,438	0,661
κ SD	0,057	0,063	0,054
ρ_E	1,42	1,26	2,08
ρ_T	1,31	1,39	1,82
θ	0,54	0,52	0,57

Табл. 2 дає центральний числовий доказ на користь побудованої технології. Integrated model досягає $F1 = 0,742$, тоді як log-only baseline має $F1 = 0,657$, а metrics-only baseline — $F1 = 0,593$. Отже, приріст integrated model становить 0,085 відносно log-only і 0,149 відносно metrics-only. Ці значення є достатньо великими, щоб їх не можна було пояснити лише випадковою варіацією сплітів. Другий характеристичний результат табл. 2 пов'язаний з ROC-AUC. Integrated model досягає 0,892, тоді як log-only і metrics-only залишаються на рівнях 0,824 і 0,781. Виграш у 0,068 проти log-only і 0,111 проти metrics-only означає, що перевага integrated model не обмежується локальною оптимізацією одного порога θ , а зберігається на всій шкалі рішень. Третя числова опора — це κ . Її значення дорівнює 0,661 для integrated model, 0,517 для log-only і 0,438 для metrics-only. Саме цей розрив показує, що integrated model не просто частіше «вгадує» позитивний клас, а й істотно краще узгоджується з еталонною розміткою. Прикладні показники посилюють цей висновок. Значення $\rho_E = 2,08$ означає більше двократного скорочення частки випадків, які передаються на ручний аналіз, тоді як $\rho_T = 1,82$ означає майже 82 %-й виграш у часовій ефективності порівняно з базовим сценарієм.

Зафіксований вмістом табл. 2 факт переваги integrated model ще не пояснює, які саме компоненти ознакового простору забезпечують цей виграш. Для цього необхідно перевірити, як модель

поводиться після вилучення окремих блоків Ψ . Така перевірка є критичною, оскільки лише вона дозволяє відокремити справжній синергетичний ефект від простої надмірності ознак.

Таблиця 3

Абляційний аналіз впливу блоків ознакового простору

Метрика	Full	No inv	No log	No metric	No dynamic
$F1$ -score	0,742	0,703	0,676	0,716	0,697
$\Delta F1$	0,000	-0,039	-0,066	-0,026	-0,045
ROC -AUC	0,892	0,861	0,842	0,873	0,856
ΔROC -AUC	0,000	-0,031	-0,050	-0,019	-0,036
Cohen's κ	0,661	0,602	0,566	0,624	0,591
$\Delta \kappa$	0,000	-0,059	-0,095	-0,037	-0,070
ρ_E	2,08	1,76	1,59	1,84	1,71
ρ_T	1,82	1,68	1,61	1,55	1,49

Табл. 3 показує, що найбільше погіршується якість після вилучення логового блоку. У цьому випадку $F1$ знижується з 0,742 до 0,676, тобто на 0,066, а κ — з 0,661 до 0,566, тобто на 0,095. Це найбільші втрати серед усіх абляцій, і саме вони вказують, що $\Psi_{\log}$ є головним носієм локальної інформації про перехід у дефектний режим. Водночас вилучення інваріантного блоку також дає системний спад. $F1$ зменшується до 0,703, тобто на 0,039, а κ — до 0,602, тобто на 0,059. Ці втрати менші ніж для $\Psi_{\log}$, але вони важливі методично, бо показують, що формально орієнтований компонент не є декоративним доповненням до емпіричних ознак, а реально стабілізує рішення моделі. Менший, але все одно ненульовий ефект дає вилучення метричного та динамічного блоків. Для Ψ_{met} спад $F1$ становить 0,026, а для Ψ_{dyn} — 0,045. Це означає, що ці компоненти не замінюють логові сигнали, але зменшують невизначеність на межі між безпечними й дефектними режимами.

Після числового зіставлення конфігурацій необхідно перейти до графічної перевірки того, чи зберігається перевага integrated model на всьому діапазоні рішень, а не лише в точці, де обрано оптимальний θ . Саме це й фіксує ROC-аналіз.

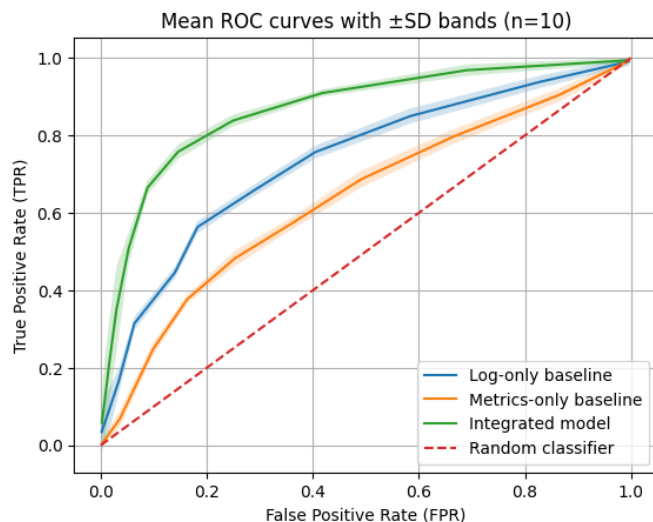


Рис. 2. Середні ROC-криві зі смугами стандартного відхилення

що metrics-only baseline виявляється не лише слабшою за integrated model, а й систематично гіршою за log-only baseline. Це безпосередньо підтверджує тезу, що самі метрики фіксують деградацію режиму менш локально й менш семантично визначено, ніж логові патерни виконання.

Проте навіть сильний ROC-профіль ще не дає відповіді на питання про стійкість моделі на рівні окремих повторів. Для цього потрібно розглянути розподіл $F1$, оскільки саме ця метрика найчутливіша до балансу між точністю P і повнотою R в умовах дисбалансованої вибірки.

Рис. 3 показує, що integrated model має не лише вищий середній $F1$, а й істотно кращу повторюваність. Її медіанне значення становить приблизно 0,728, тоді як для log-only воно залишається близьким до 0,638, а для metrics-only — до 0,584. Отже, навіть типовий, а не найкращий повтор

Рис. 2 показує, що integrated model домінує над обома baseline-конфігураціями практично на всій шкалі FPR . Особливо важливим є той факт, що відрив зберігається в області низьких і середніх FPR , тобто там, де система ще не «розраховується» за підвищену чутливість надмірною кількістю хибнопозитивних рішень. Це добре узгоджується з числом ROC -AUC = 0,892 з табл. 2, тоді як для log-only і metrics-only відповідні значення 0,824 і 0,781 графічно проявляються як систематично нижчі криві. Друга характеристична ознака рис. 2 — відносна вузькість смуг стандартного відхилення для integrated model. Це означає, що виграш не є наслідком одного чи двох вдаливих повторів, а зберігається в серії з 10 незалежних розбиттів. Третій важливий момент полягає в тому,

integrated model перевищує типовий повтор обох baseline-конфігурацій. Другий характеристичний результат полягає у ширині розподілу. Для integrated model інтерквартильний розмах є вужчим, тоді як у metrics-only спостерігається більший розкид з провалами до 0,467. Це пояснює, чому саме metrics-only дає найнижчі значення κ і ρ_E в табл. 2. Третій момент пов'язаний із log-only baseline. Хоча вона істотно краща за metrics-only, її нижні значення все одно спускаються до 0,543, що показує вразливість до окремих розбиттів і ще раз підкреслює роль додаткових інваріантних, метричних і динамічних компонент у стабілізованій integrated model.

Після підтвердження зовнішньої якості та повторюваності потрібно показати, що перевага integrated model має не лише статистичний, а й геометричний сенс у просторі ознак. Саме тому далі розглядається двовимірна проєкція $\mathcal{X}$, яка дозволяє оцінити ступінь відокремлення безпечних і дефектних сегментів.

Рис. 4 показує, що безпечні та дефектні сегменти формують дві просторово відмінні конфігурації, а не хаотично змішані хмари точок. Кількісне підтвердження цього дає silhouette, значення якого становить приблизно 0,57. Це вже не слабе, але й не граничне розділення. Саме така величина добре узгоджується з $F1 = 0,742$ і $\kappa = 0,661$ з табл. 2. Якби розділення було майже повним, модель мала б значно вищі значення цих метрик. Натомість silhouette на рівні близько 0,57 означає, що кластери помітно відокремлені, але між ними лишається прикордонна зона, яка й породжує частину FP та FN . Другий важливий аспект полягає в тому, що нормальні сегменти утворюють щільнішу й компактнішу область, тоді як дефектні сегменти мають ширший розкид. Це добре узгоджується з ідеєю, що Γ_{safe} задає стабільні режими виконання, тоді як Γ_{err} охоплює кілька різномірних сценаріїв помилки.

Останній крок емпіричної верифікації повинен показати, які саме блоки та ознаки формують виявлене геометричне розділення. Без цього перевага integrated model залишалася б статистичною, але не інтерпретованою.

Рис. 5 показує, що найбільшу концентрацію інформативності мають інваріантні та логові ознаки. Це узгоджується з табл. 3, де саме вилучення Ψ_{log} дало найбільше падіння $F1$ на 0,066 і κ на 0,095, а вилучення Ψ_{inv} — системний, хоча й менший спад $F1$ на 0,039. Відповідно, теплокарта не лише доводить доречність, а чисельно підтверджує абляційний аналіз.

Другий характеристичний результат полягає в тому, що метричні та динамічні ознаки мають нижчу, але стабільно ненульову вагу. Саме це пояснює, чому їх вилучення не руйнує модель повністю, але послідовно зменшує $F1$, $ROC-AUC$ і κ . Третій момент пов'язаний з відсутністю абсолютного домінування одного блоку. Навіть найсильніші логові й інваріантні ознаки не роблять metric- і dynamic-блоки зайвими.

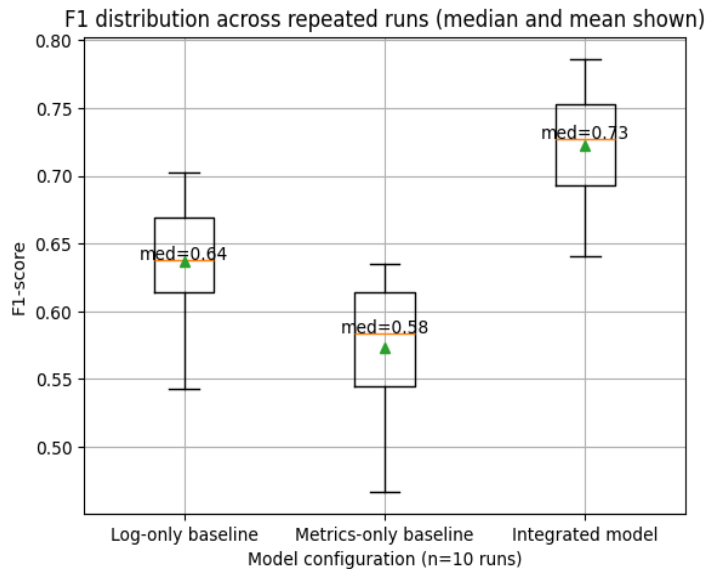


Рис. 3. Розподіл значень $F1$ -міри за повторними запусками

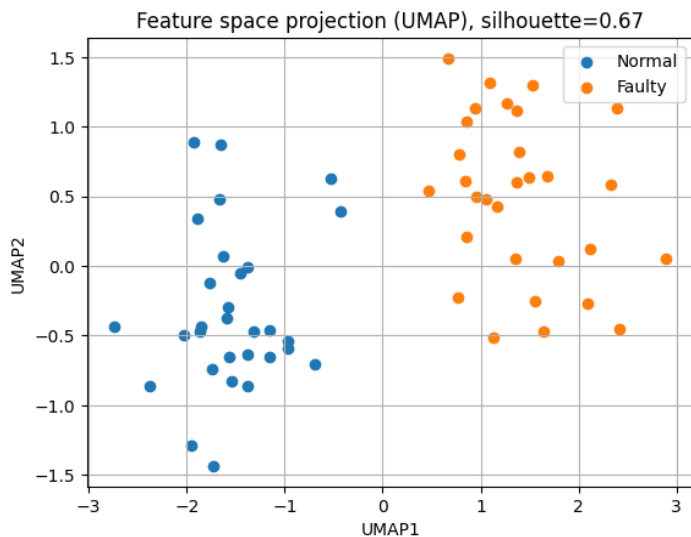


Рис. 4. Проєкція ознакового простору методом UMAP

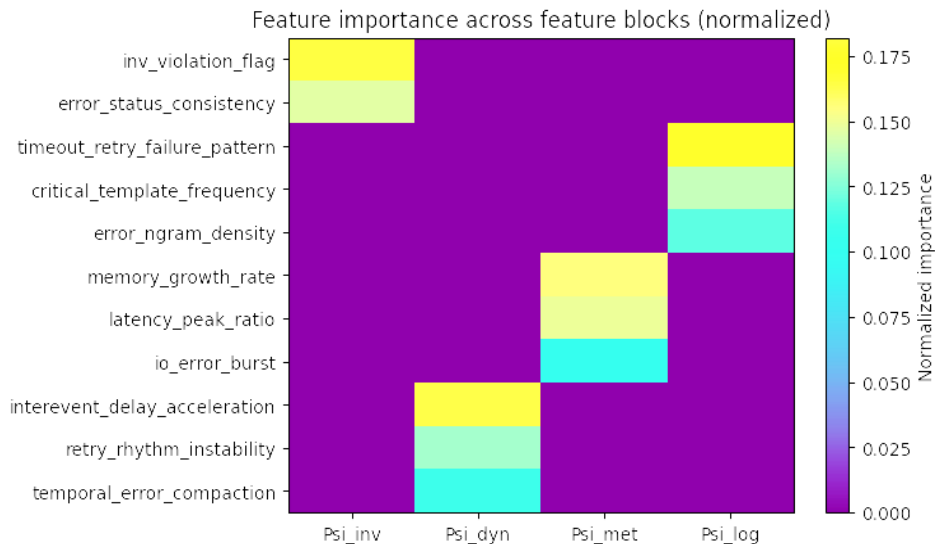


Рис. 5. Теплокарта важливості ознак за блоками ознакового простору

Висновки

Актуальність дослідження визначається не лише загальною потребою в підвищенні ефективності автоматизованого аналізу програмного забезпечення, а насамперед тим, що між формальними підходами до виявлення помилок і методологією машинного навчання досі зберігається методичний розрив. Формальний аналіз забезпечує інтерпретованість, але часто виявляється неповним або обчислювально надмірним у складних середовищах виконання. Натомість log-only і metrics-only моделі здатні виявляти поведінкові відхилення, але не дають достатньої узгодженості з формально визначеними властивостями виконання. Саме подолання цього розриву і становить безпосередню наукову мотивацію роботи.

У статті запропоновано інформаційну технологію автоматичного виявлення програмних помилок, у якій логи виконання інтерпретуються як формалізована проєкція траєкторій виконання програм, а методи машинного навчання використовуються не як автономний інструмент класифікації, а як засіб апроксимації формально визначених властивостей виконання в межах єдиної інтегрованої схеми «статичні сигнали – модель виконання – логи/метрики – ознаковий простір – поведінкова модель – індукція рішень». На відміну від підходів, що працюють або лише з журналами подій, або лише з метриками, запропонована побудова забезпечує спільне використання інваріантних, логових, метричних і динамічних компонент для прийняття рішення про належність траєкторії до дефектного режиму.

У межах проведеної експериментальної верифікації на відкритому датасеті LO2 інтегрована модель як емпірична реалізація запропонованої інформаційної технології перевищила обидві базові конфігурації за всіма ключовими критеріями. Значення $F1 = 0,742$ перевищує показники лог-орієнтованої базової моделі і метрики-орієнтованої базової моделі на 0,085 і 0,149 відповідно, що вказує не на випадкове покращення, а на структурно кращий баланс між точністю і повнотою в умовах дисбалансованої вибірки. Значення $ROC-AUC = 0,892$ проти 0,824 і 0,781 свідчить, що інтегрована модель краще розділяє безпечні й дефектні режими не лише в точці обраного порога, а на всій шкалі рішень. Коефіцієнт узгодженості $\kappa = 0,661$ проти 0,517 і 0,438 показує, що виграш пов'язаний не просто зі зростанням кількості істинно-додатних спрацьовувань, а з реально кращою відповідністю автоматизованих рішень еталонній розмітці. Практичний ефект також виявився суттєвим. Значення $\rho_E = 2,08$ означає більш ніж двократне скорочення частки випадків, що потребують ручного аналізу, а $\rho_T = 1,82$ відображає істотне зменшення часових витрат на опрацювання одного випадку. Додатково абляційний аналіз показав, що найбільший спад $F1$ спостерігається у разі вилучення логового блоку і становить 0,066, тоді як вилучення інваріантного блоку знижує $F1$ на 0,039. Це означає, що перевага інтегрованої моделі не є наслідком механічного об'єднання ознак, а формується через змістовну взаємодію логових і формально орієнтованих компонент. Наведені кількісні оцінки характеризують результати саме експерименту на LO2. Можливість перенесення встановлених закономірностей на інші програмні системи розглядається як узагальнення, що потребує окремої

емпіричної перевірки та калібрування.

Практичне значення отриманих результатів полягає в тому, що запропонована інформаційна технологія може бути використана в середовищах автоматизованого моніторингу й аналізу програмного забезпечення, насамперед у мікросервісних системах, де логи й метрики є природним джерелом спостережень, а ручне опрацювання інцидентів є дорогим і повільним. Її застосування дає змогу не лише підвищити якість автоматичного виявлення дефектних режимів, а й суттєво скоротити навантаження на інженера-експерта в процесах triage, первинної діагностики та пріоритизації інцидентів. У цьому сенсі практичний ефект технології полягає не лише в кращих числових метриках, а у переведенні частини аналітичної роботи з ручного в автоматизований режим без істотної втрати надійності.

Водночас результати мають низку обмежень. По-перше, верифікація виконувалася на відкритому датасеті, у якому внутрішні стани виконання програми не спостерігаються безпосередньо, тому частина формально визначених властивостей відновлюється через проксі-спостереження в логах і метриках. По-друге, якість моделі істотно залежить від якості журналювання і від повноти метричних спостережень. Неповні, нерівномірні або семантично збіднені логи зменшують інформативність $\Psi_{\log}$, а пропуски в метриках послаблюють внесок Ψ_{met} і Ψ_{dyn} . По-третє, результати залежать від параметризації сегментації, зокрема від вибору довжини вікна h . Це означає, що перенесення технології на інші програмні системи потребує окремої калібровки, а не може розглядатися як повністю безпараметричне.

Подальші дослідження доцільно спрямувати на застосуванні складніших моделей для апроксимації дефектної області, зокрема графових і послідовнісних архітектур, але не як самоцілі, а як засобу збереження багаторівневої структури схеми «траєкторія – проєкція – сегмент – ознаковий простір – рішення». Саме така логіка дає змогу розвивати запропоновану інформаційну технологію без втрати її формальної інтерпретованості.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

- [1] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1285-1298, 2017. <https://doi.org/10.1145/3133956.3134015>.
- [2] W. Meng, et al., "LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs," *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pp. 4739-4745, 2019. <https://doi.org/10.24963/ijcai.2019/658>.
- [3] X. Zhang, et al., "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 807-817, 2019. <https://doi.org/10.1145/3338906.3338931>.
- [4] S. Hashemi, and M. Mäntylä, "OneLog: towards end-to-end software log anomaly detection," *Automated Software Engineering*, vol. 31, no. 2, 2024. <https://doi.org/10.1007/s10515-024-00428-x>.
- [5] S. Chen, and H. Liao, "BERT-Log: Anomaly Detection for System Logs Based on Pre-trained Language Model," *Applied Artificial Intelligence*, vol. 36, no. 1, 2022. <https://doi.org/10.1080/08839514.2022.2145642>.
- [6] S. Zhang, et al., "End-to-End AutoML for Unsupervised Log Anomaly Detection," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024. <https://doi.org/10.1145/3691620.3695535>.
- [7] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "USAD," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3395-3404, 2020. <https://doi.org/10.1145/3394486.3403392>.
- [8] S. Tuli, G. Casale, and N. R. Jennings, "TranAD," *Proceedings of the VLDB Endowment*, vol. 15, no. 6, pp. 1201-1214, 2022. <https://doi.org/10.14778/3514061.3514067>.
- [9] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, and S.-K. Ng, "MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks," *Lecture Notes in Computer Science*, pp. 703-716, 2019. https://doi.org/10.1007/978-3-030-30490-4_56.
- [10] A. Aziz, and K. Munir, "Anomaly Detection in Logs Using Deep Learning," *IEEE Access*, vol. 12, pp. 176124-176135, 2024. <https://doi.org/10.1109/access.2024.3506332>.
- [11] S. He, T. Deng, B. Chen, R. Simon Sherratt, and J. Wang, "Unsupervised Log Anomaly Detection Method Based on Multi-Feature," *Computers, Materials & Continua*, vol. 76, no. 1, pp. 517-541, 2023. <https://doi.org/10.32604/cmc.2023.037392>.

Рекомендована кафедрою комп'ютерних систем управління ВНТУ

Дата надходження 23.04.2026

Дата прийняття після доопрацювання 15.06.2026

Дата прийняття до друку після рецензування 4.09.2026

Дата публікації 2.10.2026

Гуральник Фредерік Борисович — аспірант кафедри комп'ютерних систем управління, e-mail: frederikguralnik@gmail.com . <https://orcid.org/0009-0009-3832-0206>;

Ковтун В'ячеслав Васильович — д-р техн. наук, професор, завідувач кафедри комп'ютерних систем управління, e-mail: kovtun_v_v@vntu.edu.ua . <https://orcid.org/0000-0002-7624-7072>.

Вінницький національний технічний університет, Вінниця

F. B. Huralnyk¹

V. V. Kovtun¹

Information Technology for Automated Detection of Software Faults Based on Formalized Projection of Execution Trajectories and Integration of Logs and Metrics

¹Vinnytsia National Technical University

This paper investigates the problem of automated software faults detection under conditions where formal analysis ensures interpretability but is not always sufficiently complete or computationally feasible, while log-only and metrics-only approaches capture behavioral deviations but do not guarantee consistency with formally specified execution properties.

For the first time, an information technology for automated software faults detection is proposed, in which execution logs are interpreted as a formalized projection of program execution trajectories, and machine learning methods are used to approximate execution properties defined within an integrated framework of "static signals – execution model – logs/metrics – feature space – behavioral model – decision induction".

The proposed approach integrates invariant, log-based, metric, and dynamic features and transforms the problem from isolated analysis of event logs or metric time series into a task of consistent recognition of defective execution modes.

Experimental verification was conducted on the open LO2 dataset, which combines event logs and execution metrics in a microservices environment. The obtained results demonstrate that the integrated model, as an implementation of the proposed information technology, achieves an F1-score of 0.742, outperforming the corresponding log-based baseline model (0.657) and the metrics-based baseline model (0.593).

The area under the ROC curve is 0.892, indicating higher overall discriminative capability of the model in distinguishing between normal and defective execution modes. Cohen's kappa coefficient equals 0.661, confirming better agreement between automated decisions and the reference annotations.

The practical effect is manifested in more than a twofold reduction in the proportion of cases requiring expert interpretation, as well as in a significant decrease in the analysis time per case.

An ablation study further shows that log-based and invariant components contribute the most to detection performance, while metric and dynamic features serve a stabilizing and refining role.

Keywords: automated software testing, information technology, log analysis, quality metrics, machine learning, anomaly detection, behavioral modeling, microservices architecture.

Huralnyk Frederik B. — Post-Graduate Student of the Chair of Computer Control Systems, e-mail: frederikguralnik@gmail.com . <https://orcid.org/0009-0009-3832-0206>;

Kovtun Viacheslav V. — Dr Sc. (Eng.), Professor, Head of the Chair of Computer Control Systems, e-mail: kovtun_v_v@vntu.edu.ua . <https://orcid.org/0000-0002-7624-7072>