НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
В.о. завідувача кафедри
_____ Артем ВОЛОКИТА
(підпис)
“__” _____2026 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою “Комп’ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія”

на тему: Система збору та аналізу сигналів безпеки комп'ютерної інфраструктури

Виконав: студент 4 курсу, групи ІО-23

Боднар А. А. _____
(підпис)

Керівник: д.т.н. проф. Сергієнко А. М. _____
(підпис)

Консультант (нормоконтроль): ас., д-р. філос. Коренко Д. В. _____
(підпис)

Рецензент: асистент д-р. філос. Молчанов О. А. _____
(підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2026 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Артем ВОЛОКИТА
(підпис)
“__” _____2026 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Боднара Андрія Анатолійовича

1. Тема проєкту *Система збору та аналізу сигналів безпеки комп'ютерної інфраструктури*
керівник проєкту Сергієнко Анатолій Михайлович, д.т.н., проф.
затверджені наказом по університету від *03.06.2026 року №2055-с*
2. Термін здачі студентом закінченого проєкту *1 червня 2026 р.*
3. Вихідні дані до проєкту *технічна документація, теоретичні та статистичні дані*
4. Зміст розрахунково-пояснювальної записки
Системи і методи моніторингу безпеки та виявлення аномалій комп'ютерної інфраструктури. Алгоритми і технології для збору та аналізу сигналів безпеки. Система збору та аналізу сигналів безпеки. Випробування розробленої системи.

5. Перелік графічного матеріалу: блок-схема алгоритму, схема функціональна, схема структурна.
6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Коренко Д. В.		

7. Дата видачі завдання «15» січня 2026 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*03.02.2026-10.02.2026*	
2.	*Вивчення та аналіз завдання*	*10.02.2026-24.02.2026*	
3.	*Розробка архітектури та загальної структури системи*	*24.02.2026-14.03.2026*	
4.	*Розробка структур окремих підсистем*	*14.03.2026-28.03.2026*	
5.	*Програмна реалізація системи*	*28.03.2026-25.04.2026*	
6.	*Оформлення пояснювальної записки*	*25.04.2026-31.05.2026*	
7.	*Передзахист*	*08.06.2026*	
8.	*Захист*	*15.06.2026*	

Студент-дипломник _____ Андрій БОДНАР
(підпис)

Керівник проєкту _____ Анатолій СЕРГІЄНКО
(підпис)

Анотація

У роботі розглянуто методи виявлення аномалій та сучасні системи моніторингу безпеки комп'ютерної інфраструктури, проаналізовано їхні переваги й недоліки. На основі аналізу обрано гібридну трирівневу архітектуру, що поєднує правиловий двигун, ML-модель та шар великих мовних моделей.

Розроблено серверний програмний комплекс, який у реальному часі приймає події безпеки, виявляє аномальну поведінку та формує природномовну діагностику інцидентів з посиланнями на схожі історичні випадки. Реалізовано чотири взаємозамінні рушії AI-діагностики та виконано дистиляцію знань у локальну модель. Програмний продукт розроблено мовою Python.

Ключові слова: моніторинг інфраструктури, виявлення аномалій, гібридна архітектура, великі мовні моделі, дистиляція знань, Python.

Annotation

This project examines anomaly detection methods and modern computer infrastructure security monitoring systems, analyzing their strengths and weaknesses. Based on the analysis, a hybrid three-tier architecture combining a rule engine, an ML model, and a large language model layer was selected.

A server software complex was developed that receives security events in real time, detects anomalous behavior, and produces natural-language diagnostics of incidents with references to similar historical cases. Four interchangeable AI diagnostic engines were implemented, and knowledge distillation into a local model was performed. The software product was implemented in Python.

Keywords: infrastructure monitoring, anomaly detection, hybrid architecture, large language models, knowledge distillation, Python.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземп-ляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Система збору та аналізу	7		
			сигналів безпеки			
			комп'ютерної інфраструктури			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Пояснювальна записка	98		
	А4	***ІАЛЦ.467200.004 Д1***	Блок-схема алгоритму	1		
	А4	***ІАЛЦ.467200.005 Д2***	Схема функціональна	1		
	А4	***ІАЛЦ.467200.006 Д3***	Схема структурна	1		
	А4	***ІАЛЦ.467200.007 Д4***	Текст програми	1		

					ІАЛЦ.467200.001 ОА	
Зм.	Лист.	№ докум.	Підп.	Дата		
Розробив		Боднар А. А.			**Система збору та аналізу сигналів безпеки комп'ютерної інфраструктури** Опис альбому	Літ. / Аркуш: 1 / Аркушів: 1
Перевірив		Сергієнко А. М.				КПІ ім. Ігоря Сікорського ФІОТ ІО-23
Реценз.		Молчанов О. А.				
Н. Контр.		Коренко Д. В.				
Затверд.		Волокита А. М.				

ТЕХНІЧНЕ ЗАВДАННЯ ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Система збору та аналізу сигналів безпеки комп'ютерної інфраструктури»

Київ — 2026 р.

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

Найменування розробки — «Система збору та аналізу сигналів безпеки комп’ютерної інфраструктури».

Розробка являє собою серверний програмний комплекс, що в режимі реального часу приймає уніфікований потік подій з різнорідних джерел (журнали застосунків, мережеві сповіщення, повідомлення про збої розгортань, маркери атак), виявляє аномальну поведінку гібридним методом, який поєднує правиловий двигун, статистичні методи машинного навчання та природномовну інтерпретацію, та формує діагностику інцидентів з посиланнями на схожі історичні випадки.

Область застосування — моніторинг безпеки комп’ютерної інфраструктури в малих та середніх організаціях, центри реагування на інциденти, команди розробки та супроводу, які потребують уніфікованої точки спостереження за подіями безпеки з інтерпретованою діагностикою.

2. ПРИЧИНИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломний проєкт на здобуття ступеня бакалавра за освітньо-професійною програмою «Комп’ютерні системи та мережі» спеціальності 123 «Комп’ютерна інженерія», затверджене кафедрою обчислювальної техніки факультету інформатики та обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є автоматизація процесу виявлення аномальної поведінки в потоці подій безпеки комп'ютерної інфраструктури та надання оператору інтерпретованої природномовної діагностики інциденту з рекомендованими діями.

Призначення розробки — забезпечення:

- уніфікованого прийому подій з різних джерел через мережевий програмний інтерфейс;
- детекції критичних патернів з малим часом відгуку засобами детермінованого правилового двигуна;
- статистичного виявлення аномалій з урахуванням індивідуального профілю кожної підсистеми;
- природномовної діагностики з посиланнями на схожі історичні інциденти;
- можливості локального інференсу для забезпечення приватності даних організації;
- візуалізації потоку подій та результатів аналізу через інтерактивний дашборд.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література в галузях моніторингу безпеки інформаційних систем, виявлення аномалій методами машинного навчання, систем класу SIEM (Security Information and Event Management), технологій великих мовних моделей та архітектури мережевих застосунків. Використано офіційну документацію відкритих програмних бібліотек та фреймворків, публікації у періодичних виданнях, матеріали галузевих конференцій.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до продукту, що розробляється

Програмний продукт повинен реалізовувати такі функції:

- прийом подій безпеки через мережевий програмний інтерфейс із валідацією схеми вхідних даних;
- збереження подій у реляційному сховищі з підтримкою багатоарендної моделі через ідентифікатор підсистеми;
- детермінований правиловий двигун із реалізацією щонайменше чотирьох правил для виявлення критичних подій, сплесків спроб атак, серій невдалих розгортань та сплесків помилок;
- розрахунок індивідуального профілю нормальної активності кожної підсистеми з агрегацією за днями тижня;
- детектор аномалій на основі алгоритму машинного навчання класу unsupervised learning з виходом у вигляді нормалізованого показника аномальності;
- класифікатори типу атаки та рівня критичності з використанням балансування класів для роботи з незбалансованими наборами даних;
- пошук схожих історичних інцидентів у векторному сховищі через косинусну подібність векторних представлень;
- формування природномовної діагностики у структурованому форматі (стан системи, ймовірна першопричина, рекомендовані дії);
- багаторівневий захист шару діагностики від атак типу prompt-injection;
- візуалізація результатів через веб-дашборд оператора з відображенням часової шкали подій, накладеного профілю нормальної активності та згенерованої діагностики.

Програмний продукт повинен забезпечувати безперервну роботу серверної частини в режимі цілодобової експлуатації. Час відновлення після відмови — не більше 5 хвилин за рахунок автоматичного перезапуску сервісів хмарним провайдером.

Помилки обробки окремої події не повинні впливати на роботу системи в цілому. Усі помилки повинні логуватися у структурованому форматі.

Усі мережеві програмні інтерфейси серверної частини повинні бути захищені автентифікацією на основі попередньо узгодженого ключа доступу з порівнянням у режимі сталого часу для запобігання атакам за часом виконання.

Для шару природномовної діагностики передбачити багаторівневий контроль витрат: обмеження бюджету на рівні процесу, місячне обмеження на стороні постачальника зовнішнього сервісу та бюджетне сповіщення на стороні хмарного провайдера.

Для забезпечення приватності даних повинна бути реалізована можливість повного локального інференсу діагностики без надсилання вмісту подій за межі периметру організації.

5.2 Вимоги до програмного забезпечення

Серверна частина реалізується мовою програмування Python з використанням сучасного асинхронного веб-фреймворку, бібліотек об’єктно-реляційного відображення, бібліотек машинного навчання та векторних обчислень.

Для зберігання даних використовується реляційна система управління базами даних (СУБД) з підтримкою транзакцій.

Векторне сховище реалізується через спеціалізовану вбудовану базу даних з підтримкою індексування за косинусною подібністю.

Дашборд оператора реалізується на основі веб-фреймворку для побудови інтерактивних інтерфейсів даних, з візуалізацією через бібліотеку інтерактивних графіків.

Формат обміну даними між клієнтом та сервером — JSON через протокол HTTPS.

Серверна частина експлуатується у хмарному середовищі під керуванням ОС Linux. Клієнтська частина — у будь-якому сучасному веб-браузері (Chrome, Firefox, Edge, Safari) з підтримкою HTML5 та JavaScript.

Контейнер локального інференсу повинен підтримувати режим автоматичного масштабування до нуля реплік за відсутності запитів та автоматичного підняття за наявності навантаження.

5.3 Вимоги до апаратної частини

Мінімальні вимоги до серверної частини:

- процесор: не менше 2 віртуальних ядер;
- оперативна пам’ять: не менше 2 ГБ;
- дисковий простір: не менше 5 ГБ для бази даних та векторного сховища;
- підключення до мережі Інтернет зі стабільною пропускною здатністю не менше 10 Мбіт/с.

Мінімальні вимоги до сервера локального інференсу:

- процесор: не менше 4 віртуальних ядер;
- оперативна пам’ять: не менше 16 ГБ;
- дисковий простір: не менше 8 ГБ.

Мінімальні вимоги до клієнтського робочого місця:

- веб-браузер з підтримкою HTML5 та JavaScript;
- роздільна здатність екрана не менше 1280×720 точок;
- стабільне підключення до мережі Інтернет.

6. ЕТАПИ РОЗРОБКИ

Розробка програмного продукту виконується в такі етапи:

Назва етапу виконання	**Термін виконання**
Затвердження теми роботи	03.02.2026–10.02.2026
Вивчення та аналіз завдання	10.02.2026–24.02.2026
Розробка архітектури та загальної структури системи	24.02.2026–14.03.2026
Розробка структур окремих частин системи	14.03.2026–28.03.2026
Програмна реалізація системи	28.03.2026–25.04.2026
Виправлення помилок	25.04.2026–30.04.2026
Оформлення пояснювальної записки	30.04.2026–30.05.2026

ПОЯСНЮВАЛЬНА ЗАПИСКА

на тему: «Система збору та аналізу сигналів безпеки комп'ютерної інфраструктури»

Київ — 2026 р.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

AI	Artificial Intelligence (штучний інтелект)
API	Application Programming Interface (програмний інтерфейс прикладного застосування)
BDP	Bodnar Diploma Project (внутрішній скорочений ідентифікатор системи у заголовках та змінних оточення)
CPU	Central Processing Unit (центральний процесор)
CV	Cross-Validation (перехресна валідація моделі)
F1	F1-score (гармонічне середнє точності та повноти)
HDFS	Hadoop Distributed File System (розподілена файлова система Hadoop)
HIDS	Host-based Intrusion Detection System (хост-орієнтована система виявлення вторгнень)
HTTP/HTTPS	Hypertext Transfer Protocol (Secure) (протокол передавання гіпертексту)
IDS	Intrusion Detection System (система виявлення вторгнень)
JSON	JavaScript Object Notation (текстовий формат обміну даними)
LLM	Large Language Model (велика мовна модель)
LOF	Local Outlier Factor (локальний фактор аномальності)
LoRA	Low-Rank Adaptation (метод параметрично-ефективного донавчання моделей)
LR	Logistic Regression (логістична регресія)

ML	Machine Learning (машинне навчання)
NF4	Normalised Float 4-bit (формат 4-бітної квантизації ваг)
ORM	Object-Relational Mapping (об’єктно-реляційне відображення)
PEFT	Parameter-Efficient Fine-Tuning (параметрично-ефективне донавчання)
QLoRA	Quantised LoRA (поєднання 4-бітної квантизації та LoRA-адаптерів)
RAG	Retrieval-Augmented Generation (генерація з підкріпленням пошуком)
REST	Representational State Transfer (архітектурний стиль веб-сервісів)
RF	Random Forest (ансамбль випадкових дерев)
ROC-AUC	Area Under the Receiver Operating Characteristic Curve (площа під ROC-кривою)
SaaS	Software as a Service (програмне забезпечення як послуга)
SIEM	Security Information and Event Management (системи управління подіями та інформацією безпеки)
SLA	Service Level Agreement (угода про рівень сервісу)
SMOTE	Synthetic Minority Oversampling Technique (техніка генерації синтетичних прикладів міноритарного класу)
SOC	Security Operations Center (центр оперативного реагування на кіберінциденти)
SQL	Structured Query Language (мова структурованих запитів)

SVM	Support Vector Machine (метод опорних векторів)
TF-IDF	Term Frequency — Inverse Document Frequency (числова статистика для оцінки важливості слова в документі)
UML	Unified Modeling Language (уніфікована мова моделювання)
UUID	Universally Unique Identifier (універсальний унікальний ідентифікатор)
vCPU	Virtual CPU (віртуальне процесорне ядро)
VRAM	Video Random Access Memory (оперативна пам’ять відеопроцесора)
XGB	Extreme Gradient Boosting (метод градієнтного бустингу)
XML	Extensible Markup Language (розширювана мова розмітки)
ЄСКД	Єдина система конструкторської документації
ЄСПД	Єдина система програмної документації
СУБД	Система управління базами даних

ВСТУП

Сучасна комп'ютерна інфраструктура організацій будь-якого розміру характеризується безпрецедентною кількістю джерел подій, що потенційно можуть свідчити про загрози безпеці: журнали веб-серверів, сповіщення мережевого обладнання, повідомлення про невдалі розгортання, маркери активності зловмисників, дані систем виявлення вторгнень тощо [1]. Обсяг такого потоку перевищує можливості ручного аналізу, що зумовлює потребу в автоматизованих засобах збору, нормалізації та інтерпретації сигналів безпеки.

Існуючі промислові системи класу SIEM (Security Information and Event Management) — Splunk Enterprise Security, Elastic Security, Wazuh, Datadog Security Monitoring — забезпечують потужні засоби збору і агрегації подій, проте мають ряд суттєвих недоліків: висока вартість ліцензування, прив'язка до конкретного постачальника, обмежена інтерпретованість висновків (як правило, кінцевим артефактом є лише факт спрацювання правила), а також відсутність вбудованих засобів формування природномовного пояснення першопричин інциденту [3; 4].

Активний розвиток галузі великих мовних моделей (Large Language Models, LLM) у поєднанні з підходом Retrieval-Augmented Generation (RAG) дає можливість поєднати детерміновану детекцію подій безпеки з природномовною діагностикою на основі історичного контексту [20]. Гібридна архітектура, що поєднує детермінований рушій правил, статистичну ML-модель та шар LLM-діагностики, є малодослідженою у відкритих джерелах і становить науковий інтерес [11].

Актуальність роботи зумовлена зростанням кількості та складності кіберзагроз, потребою в інтерпретованих засобах автоматичної діагностики інцидентів, високою вартістю пропрієтарних SIEM-систем та необхідністю за-

безпечення приватності оброблюваних даних безпеки на тлі вимог законодавства України про основні засади забезпечення кібербезпеки [2].

Згідно з §5 Положення про виконання дипломних проєктів кафедри обчислювальної техніки [5], розроблений у межах цієї роботи програмний продукт одночасно належить до трьох категорій: системи аналізу й обробки даних (як основної категорії, оскільки саме приймання, агрегація та класифікація подій інфраструктури є центральною функцією), системи, що базується на знаннях (через наявність шару Retrieval-Augmented Generation і дистильованої локальної моделі, що оперують засвоєними знаннями про класи інцидентів), а також, частково, системи прогнозування (через застосування алгоритму Isolation Forest, який порівнює поточну активність з очікуваним профілем і кількісно оцінює ймовірність відхилення від норми) [12].

Об’єкт дослідження — процес виявлення аномальної поведінки у потоці подій безпеки комп’ютерної інфраструктури.

Предмет дослідження — методи та програмні засоби гібридної (rule-based + ML + LLM) детекції та інтерпретації інцидентів безпеки.

Мета дипломного проєкту — розробка серверного програмного комплексу, що в режимі реального часу приймає події безпеки комп’ютерної інфраструктури з різнорідних джерел, виявляє аномальну поведінку гібридним методом (правила + ML + LLM) та формує природномовну діагностику інцидентів з посиланнями на схожі історичні випадки.

Для досягнення поставленої мети необхідно вирішити такі **задачі**:

1) проаналізувати існуючі системи моніторингу безпеки та виявити їх недоліки;
2) обрати алгоритми виявлення аномалій та класифікації подій на основі порівняльного аналізу;
3) розробити уніфіковану схему події та реляційну модель збереження даних;

4) реалізувати детермінований рушій правил для миттєвих сповіщень;
5) реалізувати ML-шар детекції аномалій з урахуванням індивідуального профілю кожної підсистеми;
6) інтегрувати векторне сховище історичних інцидентів для застосування підходу RAG;
7) реалізувати уніфікований клієнт LLM з відстеженням бюджету та чотири взаємозамінні рушії AI-діагностики;
8) виконати дистиляцію знань teacher-моделі у локальну student-модель для приватного інференсу;
9) розробити шар захисту від атак типу prompt-injection;
10) розгорнути компоненти системи у хмарних середовищах із контролем витрат;
11) розробити інтерактивний дашборд оператора;
12) провести експериментальне дослідження ефективності розроблених компонентів.

Методи дослідження. У роботі застосовано такі методи. По-перше, методи системного аналізу та порівняльного огляду — для виявлення спільних обмежень існуючих SIEM-рішень та формулювання вимог до проєкту. По-друге, методи статистичного навчання без учителя — для побудови моделі базового профілю активності кожної підсистеми та детекції аномалій алгоритмом Isolation Forest [12]. По-третє, методи навчання з учителем (логістична регресія, випадковий ліс, градієнтний бустинг XGBoost з технікою балансування SMOTE) — для класифікації типу атаки і рівня критичності події на корпусах HDFS_v1 [7] та CICIDS2017 [8]. По-четверте, методи семантичного пошуку у багатовимірних просторах ембедингів за косинусною подібністю — для шару RAG [20; 21]. По-п’яте, методи параметрично-ефективного донавчання великих мовних моделей (LoRA [24], QLoRA [25]) — для дистиляції знань комерційної моделі-вчителя у локаль-

ну модель-учня з відкритими вагами. По-шосте, методи експериментального дослідження (A/B-прогони, перехресна валідація, red-team тестування корпусом prompt-injection атак [27]) — для оцінки якості розроблених компонентів. Розробка велась за ітеративним підходом з регулярним проміжним контролем; усі експерименти репродуцируються за фіксованими random seed та задокументованими гіперпараметрами.

Наукова новизна роботи полягає у поєднанні детермінованого рушія правил, статистичної моделі Isolation Forest з індивідуальним профілем активності та чотирьох взаємозамінних рушіїв LLM-діагностики у єдиній архітектурі, що забезпечує оператору вибір між якістю діагностики, вартістю та рівнем приватності даних. Окремим внеском є експериментальне дослідження стійкості шару LLM-діагностики до prompt-injection атак на корпусі з 27 кейсів [27] та валідація методу дистиляції знань на двох незалежних A/B-прогонах [23—25], що дозволило довести узагальнюваність результату дистиляції на не бачених раніше сценаріях.

Практичне значення одержаних результатів полягає у можливості використання розробленої системи як готового програмного комплексу для моніторингу безпеки інфраструктури малих та середніх організацій, що не мають фінансових ресурсів для впровадження пропрієтарних SIEM-систем. Крім того, локальний рушій інференсу на базі дистильованої моделі забезпечує можливість роботи в умовах обмежень на передавання даних за межі периметру організації, що є істотним для державних установ, фінансових організацій та медичних систем, які підпадають під регуляторні обмеження щодо обробки персональних даних [3; 29].

Структура пояснювальної записки. Пояснювальна записка складається зі вступу, чотирьох розділів, висновків та списку використаних джерел. У першому розділі проведено огляд предметної області та існуючих SIEM-рішень, виявлено їх ключові недоліки і сформульовано вимоги до проєкту.

Другий розділ присвячено порівняльному аналізу алгоритмів виявлення аномалій, класифікаторів, технологій векторного пошуку, великих мовних моделей та технологічного стека серверної частини. У третьому розділі описано деталі реалізації серверного програмного комплексу, базу даних, чотири рушії AI-діагностики, шар захисту від prompt-injection, дашборд оператора та сценарії експлуатації. У четвертому розділі викладено результати п’яти експериментальних досліджень компонентів системи з кількісними метриками та інтерпретаційним аналізом. Бібліографічне оформлення виконано відповідно до ДСТУ 8302:2015 [31]; оформлення пояснювальної записки — відповідно до ДСТУ 3008-95 [30].

РОЗДІЛ 1
СИСТЕМИ І МЕТОДИ МОНІТОРИНГУ БЕЗПЕКИ ТА ВИЯВЛЕННЯ АНОМАЛІЙ КОМП'ЮТЕРНОЇ ІНФРАСТРУКТУРИ

1.1 Інфраструктура комп'ютерного комплексу

Комп'ютерна інфраструктура сучасної організації являє собою розподілену систему, що складається з обчислювальних вузлів, мережевого обладнання, систем зберігання даних, програмних сервісів та засобів захисту. У процесі функціонування така інфраструктура генерує безперервний потік подій — структурованих повідомлень, кожне з яких відображає певну зміну стану або дію, виконану в системі [1].

Важливою підмножиною подій інфраструктури є події безпеки — повідомлення, що потенційно можуть свідчити про загрозу чи інцидент. Серед типових джерел подій безпеки можна виділити журнали веб-серверів та застосунків, мережеві сповіщення (системи виявлення вторгнень, міжмережеві екрани), повідомлення про невдачі процесів розгортання програмного забезпечення, дані аутентифікаційних систем та маркери активності зловмисників, отримані з систем-приманок [6; 15].

Обсяг такого потоку для організації середнього розміру може досягати від десятків тисяч до мільйонів подій на добу. Очевидно, що ручний аналіз кожної події є нереалістичним, що зумовлює потребу в автоматизованих засобах збору, нормалізації, фільтрації та інтерпретації сигналів безпеки. При цьому вимоги до таких засобів істотно відрізняються від вимог до загальносистемних засобів моніторингу: критично важливими є низька латентність формування сповіщень про підозрілу активність, інтерпретованість виснов-

ків для оператора, можливість розслідування інциденту в історичному контексті схожих минулих випадків [1; 9].

Класичним архітектурним рішенням такої задачі є системи класу SIEM (Security Information and Event Management), які забезпечують централізоване збирання журналів з різних джерел, нормалізацію форматів, кореляцію подій та формування сповіщень. Архітектурно така система складається з трьох основних шарів: шару збору даних (колектори, агенти, надсилачі), шару нормалізації та збереження (парсери, індексатори, реляційні чи документо-орієнтовані сховища), шару аналізу та візуалізації (рушій кореляції, дашборд оператора) [3]. Проте сучасні вимоги до інтерпретованості висновків та контролю вартості експлуатації виходять за межі можливостей класичних SIEM-систем [10].

1.2 Задачі моніторингу безпеки

Задачі, що виникають у процесі моніторингу безпеки комп'ютерної інфраструктури, можна умовно розділити на чотири класи, що відповідають різним типам автоматизованої обробки подій [4].

Перший клас — детекція критичних подій, тобто миттєве виявлення окремих подій, що відповідають заздалегідь відомим патернам (наприклад, підозріло велика кількість невдалих спроб автентифікації за короткий проміжок часу). Такі задачі типово вирішуються детермінованими правилами і не потребують навчання моделей. Перевагою цього класу є мінімальна латентність формування сповіщення (одиниці мілісекунд) і повна інтерпретованість результату — оператор отримує посилання на конкретне правило, що спрацювало, і має змогу швидко зрозуміти причину сповіщення.

Другий клас — детекція аномальної поведінки. На відміну від першого класу, тут йдеться не про окремі події, а про відхилення агрегованих характеристик потоку від нормального профілю системи. Це задача без учителя,

що типово розв’язується статистичними методами та алгоритмами навчання без учителя [11; 12]. Перевагою цього класу є здатність виявляти невідомі заздалегідь патерни поведінки, недоліком — відсутність у вихідному сигналі моделі прямого пояснення причини спрацювання.

Третій клас — класифікація типу атаки чи рівня критичності події (supervised класифікація). Ці задачі потребують розмічених наборів даних та застосовують класичні методи машинного навчання — від логістичної регресії до ансамблевих методів градієнтного бустингу [16; 17]. Особливістю цього класу задач є необхідність періодичного перенавчання моделей при появі нових типів атак, оскільки розподіл вхідних даних з часом змінюється (concept drift).

Четвертий клас — формування інтерпретованої діагностики для оператора. Це найскладніший клас задач, що поєднує всі попередні результати у природномовне пояснення з рекомендованими діями. Сучасний підхід до цієї задачі ґрунтується на застосуванні великих мовних моделей з опціональним підкріпленням пошуком (RAG) у сховищі історичних інцидентів [20]. Особливістю цього класу є потреба в забезпеченні якості природномовного виходу — фактичної достовірності, відсутності галюцинацій, корисності рекомендацій для дій.

Для наочного представлення взаємодії цих чотирьох класів обробки на рис. 1.1 наведено узагальнену схему обробки потоку подій безпеки, що відображає послідовність застосування зазначених класів задач. На схемі показано, що кожна вхідна подія послідовно проходить через усі чотири рівні аналізу, причому пізніші рівні використовують результати, отримані на попередніх.

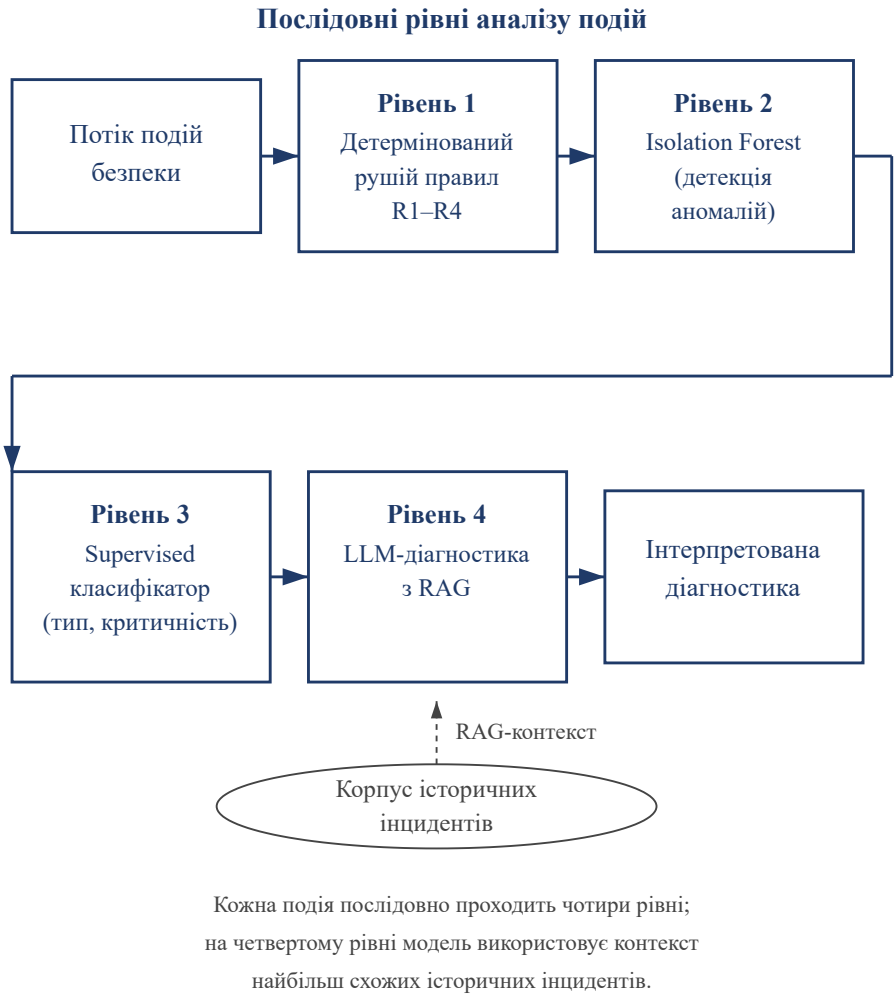


Рисунок 1.1 – Узагальнена схема обробки потоку подій безпеки

Як видно зі схеми, перший рівень (детермінований рушій правил R1–R4) забезпечує миттєве виявлення критичних патернів за заздалегідь відомими сигнатурами. Другий рівень (Isolation Forest) виявляє відхилення агрегованих характеристик потоку від нормального профілю активності системи. Третій рівень (supervised класифікатор) уточнює тип та критичність інциденту на основі наборів даних з людськими мітками. Четвертий рівень формує природномовну діагностику, додатково використовуючи корпус історичних інцидентів через шар RAG. Інтегрований вихід усіх чотирьох рівнів подається оператору як інтерпретована діагностика інциденту з рекомендованими діями. Саме така архітектурна композиція становить основу пропонованого

рішення; її конкретна реалізація для кожного рівня детально розглядається у наступних розділах.

1.3 Системи класу SIEM

Сучасний ринок SIEM-систем пропонує широкий спектр рішень — від повноцінних комерційних платформ до відкритих систем з обмеженою функціональністю [10]. У цьому підрозділі розглянуто чотири найбільш поширені рішення, що зустрічаються в інфраструктурі організацій малого та середнього розміру.

1.3.1 Splunk Enterprise Security

Splunk Enterprise Security (Splunk ES) — пропрієтарна платформа виробництва компанії Splunk Inc. Платформа побудована на основі розподіленого індексатора подій, що забезпечує приймання та збереження великих обсягів напівструктурованих даних. Корелятор подій базується на власній мові запитів SPL (Search Processing Language), яка дозволяє формулювати складні правила виявлення інцидентів.

До переваг Splunk належить розвинена екосистема розширень (Splunkbase налічує тисячі модулів), велика база готових кореляційних правил (Common Information Model), потужні засоби візуалізації та звітності. Серед недоліків слід виділити високу вартість ліцензування — модель ціноутворення базується на обсязі даних, що індексуються щодня (від кількох тисяч доларів на місяць для невеликих інсталяцій), складність розгортання власної інфраструктури індексаторів та search-head, відсутність вбудованих засобів природномовної діагностики інцидентів [10].

1.3.2 Elastic Security (ELK Stack)

Elastic Security — рішення на основі стека Elasticsearch + Logstash + Kibana, що розповсюджується за подвійною ліцензією (SSPL та Elastic License). На відміну від Splunk, базова функціональність доступна безкоштовно, що зробило цей стек поширеним вибором серед компаній, які прагнуть мати власну інфраструктуру моніторингу без значних ліцензійних відрахувань.

Серед переваг — велике активне співтовариство, потужний пошуковий рушій з підтримкою повнотекстового пошуку та структурованих запитів, інтегрований дашборд Kibana з гнучкими можливостями візуалізації, наявність ML-модулів для виявлення аномалій (доступні за платною підпискою). Серед недоліків — суттєві вимоги до апаратних ресурсів (індексатор Elasticsearch потребує значних обсягів оперативної пам'яті — рекомендовано від 32 ГБ на вузол), складність налаштування правил кореляції, обмежена функціональність безкоштовної версії порівняно з комерційною (зокрема, машинне навчання та частина SIEM-функцій доступні лише у комерційній підписці).

1.3.3 Wazuh

Wazuh — повністю відкрита SIEM-платформа з ліцензією GPL, що первісно виникла як форк проєкту OSSEC. Wazuh поєднує функції хост-орієнтованої системи виявлення вторгнень (HIDS), сканера вразливостей та централізованого журналу подій. Платформа інтегрується з Elasticsearch для збереження даних та з Kibana для візуалізації.

Серед переваг — повна відкритість коду, розвинена база сигнатур виявлення вторгнень (понад 3000 правил, що покривають типові патерни атак), наявність агента для встановлення на контрольованих вузлах, підтримка моніторингу цілісності файлів та аналізу конфігурацій. Серед недоліків — складність встановлення та налаштування, обмежена документація, відсутність

вбудованої AI-діагностики, залежність від стека Elasticsearch для повноцінного функціонування.

1.3.4 Datadog Security Monitoring

Datadog Security Monitoring — модуль безпеки у складі SaaS-платформи Datadog. На відміну від Splunk, Elastic і Wazuh, що передбачають розгортання власної інфраструктури, Datadog працює як хмарний сервіс, до якого передаються події безпеки клієнта. Це спрощує експлуатацію, але створює залежність від зовнішнього постачальника та обмежує можливості локалізації даних [29].

Серед переваг — простота інтеграції (підтримка десятків готових колекторів для різних джерел даних), інтегрований моніторинг інфраструктури та безпеки в одній платформі, регулярні оновлення правил виявлення з боку постачальника, доступ до виявлення аномалій на основі машинного навчання як частина платної підписки. Серед недоліків — висока вартість при значних обсягах даних (модель ціноутворення базується на обсязі логів, кількості хостів та контейнерів), відсутність контролю над збереженням даних та обмежена можливість локалізації даних у межах конкретної юрисдикції, що може бути критичним для організацій, що підпадають під регуляторні обмеження [2; 3].

1.3.5 Порівняльний аналіз існуючих рішень

Для системного порівняння розглянутих рішень за основними характеристиками, релевантними для задачі дипломного проєкту, у табл. 1.1 наведено узагальнений аналіз чотирьох SIEM-систем. Розгорнуту порівняльну архітектурну схему чотирьох розглянутих SIEM-систем наведено на рис. 1.2.

Таблиця 1.1 – Порівняльний аналіз існуючих SIEM-систем

Характеристика	**Splunk ES**	**Elastic Security**	**Wazuh**	**Datadog**
Тип ліцензії	Пропрієтарна	SSPL / Elastic License	GPL (відкрита)	SaaS-сервіс
Вартість для малого SOC	Висока	Помірна	Низька	Помірна
Локальне розгортання	Так	Так	Так	Ні
Природномовна діагностика	Ні	Часткова (ML)	Ні	Часткова
Підхід RAG над інцидентами	Ні	Ні	Ні	Ні
Інтерпретованість висновків	Спрацювання правила	Спрацювання правила	Спрацювання правила	Спрацювання правила
Vendor lock-in	Високий	Середній	Низький	Високий
Залежність від хмари	Опційно	Опційно	Ні	Так (повна)

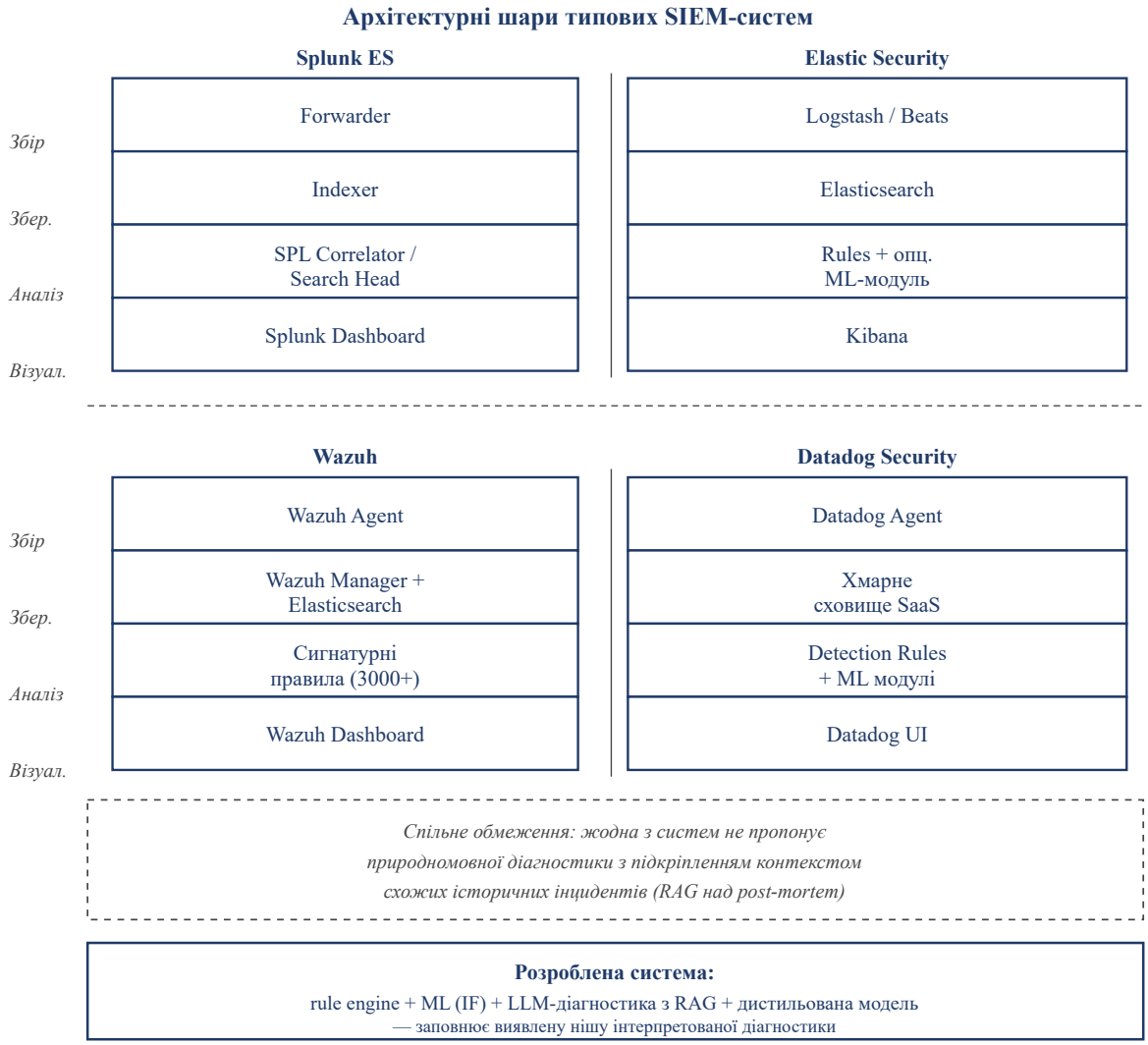


Рисунок 1.2 – Порівняльна архітектура існуючих SIEM-систем

Як видно з рис. 1.2, усі чотири розглянуті системи мають принципово схожу архітектурну композицію з чотирьох шарів — збору, збереження, аналізу та візуалізації. При цьому шар аналізу в усіх випадках обмежується виявленням спрацювань заздалегідь сформульованих правил кореляції; жодна з систем не пропонує вбудованої природномовної діагностики інцидентів з підкріпленням контекстом схожих історичних випадків. Таким чином, у даному сегменті ринку наявна функціональна ніша для гібридного рішення, що поєднує детермінований рушій правил, ML-шар і LLM-діагностику з RAG,

забезпечуючи оператору не лише факт спрацювання, а й змістовне пояснення першопричини інциденту [11; 16].

1.4 Підходи до автоматичної діагностики інцидентів

Підходи до автоматичної діагностики інцидентів безпеки історично розвивалися від простих правилових систем до сучасних AI-орієнтованих рішень. Можна виділити чотири покоління цих підходів.

Перше покоління — правилові експертні системи. Класичні системи виявлення вторгнень (наприклад, Snort, Suricata) працюють на основі сигнатур, написаних експертами вручну. Такі системи відзначаються низькою латентністю та повною інтерпретованістю висновків — для кожного спрацювання можна точно вказати правило, що його сформувало. Проте вони мають принципову обмеженість: можуть виявляти лише відомі заздалегідь патерни атак, а оновлення сигнатур відбувається з затримкою у дні-тижні після появи нових загроз [15].

Друге покоління — статистична детекція аномалій. Підходи цього класу використовують моделі без учителя (наприклад, Isolation Forest, One-Class SVM, Local Outlier Factor) для виявлення відхилень від нормального профілю [12—14]. Перевагою є здатність виявляти раніше невідомі патерни, недоліком — відсутність пояснення причини спрацювання. Оператор отримує лише числове значення оцінки аномальності, без вказівки, які саме характеристики потоку відхилилися від норми.

Третє покоління — supervised класифікація з використанням розмічених корпусів даних. На основі публічних наборів даних, таких як HDFS_v1 (Loghub) [7; 19] та CICIDS2017 [8], можна тренувати моделі, що з високою точністю класифікують події за типом атаки чи рівнем критичності [16—18]. Такі моделі забезпечують вищу точність порівняно зі сигнатурними система-

ми, але потребують періодичної переоцінки та ретренування при появі нових типів атак.

Четверте покоління — інтерпретована діагностика на основі великих мовних моделей. Останні роки характеризуються активним розвитком LLM (наприклад, моделі сімейства Claude компанії Anthropic, моделі сімейства GPT компанії OpenAI), що здатні формувати природномовний опис інциденту з рекомендованими діями. Якість таких діагнозів істотно зростає при використанні підходу Retrieval-Augmented Generation, що передбачає пошук схожих історичних випадків у векторному сховищі та подачу їх у контекст моделі [20; 21]. Окремою тенденцією четвертого покоління є застосування function-calling агентів — LLM, які ітеративно викликають набір інструментів для збору необхідної інформації перед формуванням відповіді.

Розроблена в межах дипломного проєкту система належить до четвертого покоління і поєднує всі попередні підходи в єдиній архітектурі: детермінований рушій правил для миттєвих сповіщень, статистичний ML-шар для виявлення аномалій, supervised класифікатори для визначення типу інциденту, та LLM-діагностику з RAG для формування інтерпретованого висновку.

1.5 Вимоги до проєкту

На основі проведеного огляду сформульовано такі вимоги до проєкту. Необхідно розробити серверний програмний комплекс, що задовольняє вимоги, наведені нижче, при цьому відповідаючи вимогам ДСТУ ISO/IEC 27001:2015 щодо систем управління інформаційною безпекою [3] та враховуючи рекомендації ДСТУ ISO/IEC 27035-1:2018 щодо принципів керування інцидентами інформаційної безпеки [4]. До основних функціональних вимог системи належать:

– прийом подій безпеки з різнорідних джерел через уніфікований HTTP-інтерфейс із валідацією схеми;

– збереження подій у реляційному сховищі з підтримкою мультитенантності;
– реалізація детермінованого рушія правил, що забезпечує миттєві сповіщення про критичні події;
– реалізація ML-шару виявлення аномалій, що враховує індивідуальний профіль кожної підсистеми за днями тижня;
– інтеграція векторного сховища історичних інцидентів та реалізація retrieval-augmented діагностики;
– реалізація щонайменше чотирьох взаємозамінних рушіїв LLM-діагностики, що повертають однотипну структуровану відповідь;
– дистиляція знань з teacher-моделі (комерційна LLM) у локальну student-модель для приватного інференсу;
– реалізація шару захисту від атак типу prompt-injection;
– розгортання компонентів у хмарних середовищах із багаторівневим контролем витрат;
– розробка інтерактивного дашборду оператора з візуалізацією потоку подій та результатів діагностики;
– експериментальне дослідження ефективності розроблених компонентів на відкритих наукових корпусах.

Окрім функціональних, ставляться нефункціональні вимоги: оброблення події рушієм правил має вкладатися в час миттєвого сповіщення, а оцінювання аномальності та LLM-діагностика виконуються поза трактом приймання подій; приватність забезпечує локальна дистильована модель; експлуатаційна вартість контролюється на трьох рівнях; архітектура лишається розширюваною із взаємозамінними рушіями, а відповідність перевіряється експериментально на відкритих наукових корпусах.

ВИСНОВКИ ДО РОЗДІЛУ 1

У першому розділі проведено огляд предметної області — моніторингу безпеки комп'ютерної інфраструктури. Проаналізовано чотири найпоширеніші системи класу SIEM (Splunk Enterprise Security, Elastic Security, Wazuh, Datadog Security Monitoring) та виявлено їх ключові недоліки: висока вартість, vendor lock-in, обмежена інтерпретованість висновків та відсутність вбудованої природномовної діагностики з підкріпленням контекстом схожих історичних випадків.

Класифіковано задачі моніторингу безпеки на чотири класи: детекція критичних подій, виявлення аномальної поведінки, класифікація типу атаки та формування інтерпретованої діагностики. Розглянуто еволюцію підходів до автоматичної діагностики від правилових експертних систем до сучасних рішень на основі великих мовних моделей з функцією виклику інструментів. Показано, що розроблюваний у межах дипломного проєкту програмний комплекс належить до четвертого покоління підходів і поєднує переваги усіх попередніх.

На основі проведеного аналізу сформульовано вимоги до проєкту, що передбачає розробку гібридного програмного комплексу з поєднанням детермінованого рушія правил, статистичного ML-шару та LLM-діагностики з RAG. Алгоритми та технології, що складуть основу реалізації, розглянуто у наступному розділі.

РОЗДІЛ 2
АЛГОРИТМИ І ТЕХНОЛОГІЇ ДЛЯ ЗБОРУ ТА АНАЛІЗУ СИГНАЛІВ БЕЗПЕКИ

У цьому розділі проведено детальний порівняльний аналіз алгоритмів та технологій, які потенційно можуть бути використані для реалізації поставленої задачі. На основі аналізу обґрунтовано вибір тих методів та інструментів, що склали технологічну основу розробленої системи.

2.1 Алгоритми виявлення аномалій

Виявлення аномалій (anomaly detection) — задача навчання без учителя, що полягає у визначенні точок даних, що істотно відрізняються від очікуваного розподілу [12]. У контексті моніторингу безпеки аномаліями вважаються відхилення агрегованих характеристик потоку подій від нормального профілю активності системи [15]. Розглянемо чотири найпоширеніші алгоритми виявлення аномалій.

2.1.1 Алгоритм Isolation Forest

Isolation Forest, запропонований у роботі Liu, Ting, Zhou [12], є ансамблевим методом, побудованим на ідеї ізоляції аномальних точок шляхом випадкових розщеплень. Алгоритм будує ансамбль ізолюючих дерев (iTree), де кожне дерево формується рекурсивним випадковим розщепленням простору ознак. На кожному кроці випадково обирається ознака та випадкова точка розщеплення в межах діапазону цієї ознаки в поточному підмножині. Аномалія в такому ансамблі ізолюється швидше за нормальну точку, тобто має коротшу середню довжину шляху від кореня до листя.

Формально оцінка аномалії точки x обчислюється за виразом:

$$s(x, n) = 2^{-E(h(x))/c(n)}, \tag{2.1}$$

де $E(h(x))$ — математичне сподівання довжини шляху від кореня до листя за всіма деревами ансамблю;
$c(n)$ — нормуючий коефіцієнт, що відповідає середній довжині шляху для випадкового бінарного дерева пошуку з n елементами, обчислюється як $c(n) = 2H(n-1) - 2(n-1)/n$, де $H(\cdot)$ — гармонічне число.

Значення $s(x, n)$ лежить у діапазоні $[0; 1]$; значення близькі до 1 свідчать про аномалію, близькі до 0,5 — про нормальну точку, а значення значно менші за 0,5 указують на типове перебування точки серед більшості.

Геометричну ілюстрацію принципу ізоляції наведено на рис. 2.1. Зліва показано двовимірний простір ознак з кластером нормальних точок (потребує багато розщеплень для ізоляції довільної точки) та з виокремленою аномалією A, що ізолюється за два розщеплення. Справа показано фрагмент відповідного ізолюючого дерева, де глибина розташування точки відображає кількість розщеплень, потрібних для її ізоляції.

З рис. 2.1 наочно випливає основна перевага алгоритму: аномальні точки, розташовані далеко від основного кластера, ізолюються лише декількома випадковими розщепленнями, що відображається у малій глибині h їх розташування у дереві. Нормальні точки, навпаки, потребують глибокого занурення у дерево, бо лежать поруч з багатьма іншими точками. Усереднення h за великою кількістю дерев ансамблю стабілізує оцінку та робить її стійкою до випадкового вибору ознак при побудові окремих дерев.

Перевагами алгоритму є лінійна обчислювальна складність $O(n \log n)$, відсутність припущення про конкретний розподіл даних, низькі вимоги до пам'яті (зберігається лише структура дерев, а не вихідні дані) та інтерпре-

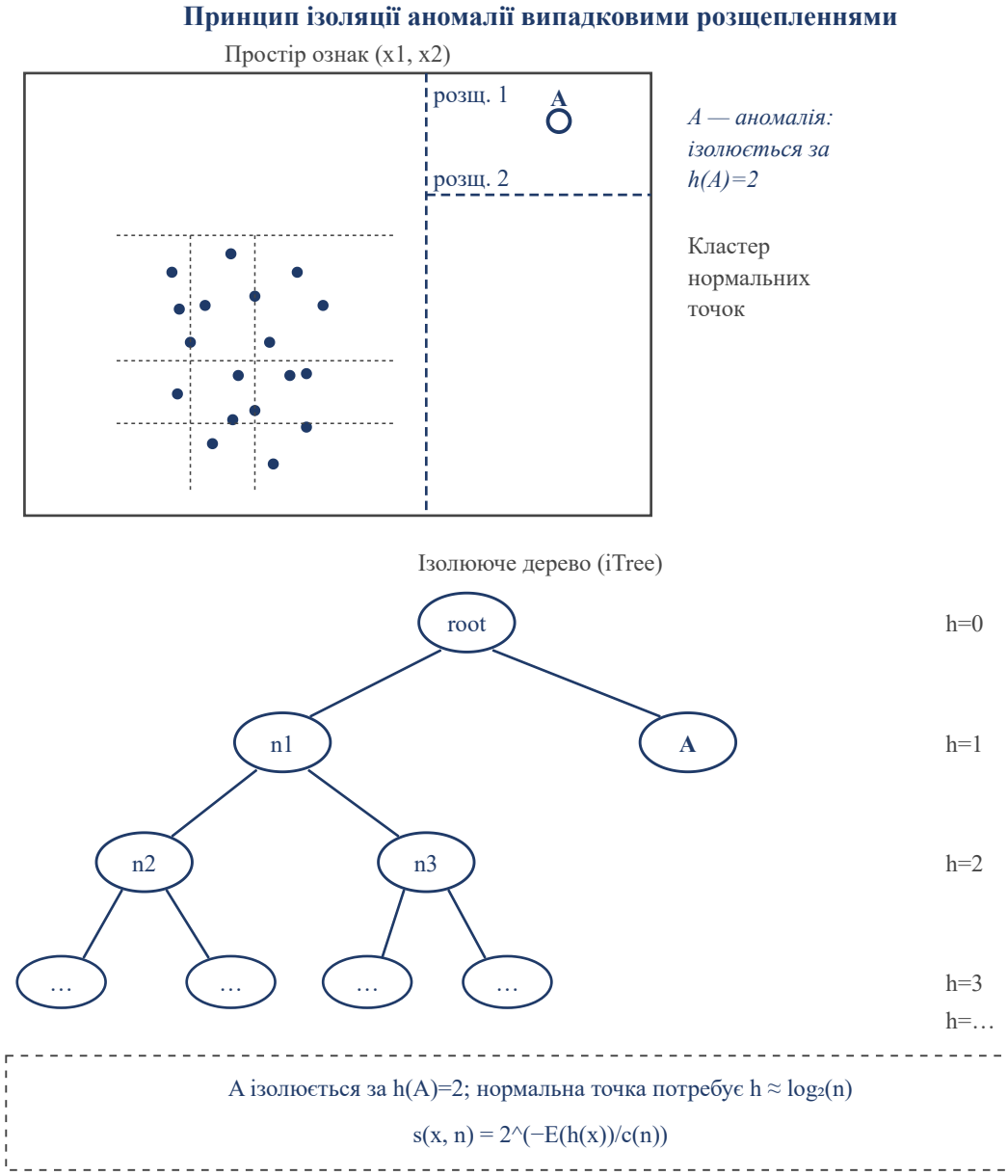


Рисунок 2.1 – Принцип ізоляції аномалії в алгоритмі Isolation Forest

тованість оцінки аномалії як ймовірнісної величини. Алгоритм добре масштабується до великих наборів даних і не потребує спеціалізованого апаратного забезпечення. Недоліком є чутливість до неправильно сформованого тренувального набору, що містить значну частку аномальних точок (понад 10–15 %) — у цьому разі дерева вчаться розпізнавати аномалії як норму.

2.1.2 Алгоритм One-Class SVM

One-Class Support Vector Machine — модифікація методу опорних векторів для задачі виявлення аномалій, запропонована у роботі Schölkopf et al. [13]. Алгоритм будує гіперплощину, що максимально відокремлює нормальні точки від початку координат у просторі ознак (з використанням ядрового перетворення, типово — радіальної базисної функції). Точки, що опиняються по інший бік гіперплощини, вважаються аномальними.

Формально задача формулюється як мінімізація:

$$\frac{1}{2}\|w\|^2 + \frac{1}{\nu n}\sum_i \xi_i - \rho \to \min, \tag{2.2}$$

за обмежень $(w \cdot \Phi(x_i)) \geq \rho - \xi_i$, $\xi_i \geq 0$,

де w — нормаль до гіперплощини;

ξ_i — slack-змінні для пом'якшення обмежень;

ρ — зміщення гіперплощини;

$\nu \in (0, 1]$ — параметр, що задає верхню межу частки точок-помилок та нижню межу частки опорних векторів.

Перевагами методу є теоретично обґрунтована постановка та можливість роботи з нелінійними залежностями через ядрові функції. Недоліками — квадратична обчислювальна складність $O(n^2)$, що ускладнює застосування на великих корпусах, та чутливість до вибору параметра ν, який визначає очікувану частку аномалій. Крім того, метод вимагає попередньої стандартизації ознак.

2.1.3 Алгоритм Local Outlier Factor

Local Outlier Factor (LOF), запропонований Breunig et al. [14], оцінює ступінь аномальності точки на основі локальної густини навколо неї порівняно з густиною її k-найближчих сусідів. Аномалія, за визначенням LOF, — це точка, локальна густина якої істотно нижча за густину сусідів.

Формально оцінка аномальності обчислюється у три кроки. По-перше, обчислюється відстань до k-го найближчого сусіда $\text{reach-dist}_k(p, q)$. По-друге, обчислюється локальна reachability density:

$$\text{lrd}_k(p) = \frac{1}{\dfrac{\sum_{q \in N_k(p)} \text{reach-dist}_k(p, q)}{|N_k(p)|}}, \tag{2.3}$$

де $N_k(p)$ — множина k найближчих сусідів.

По-третє, оцінка LOF обчислюється як середнє відношення локальних густин сусідів до густини самої точки:

$$\text{LOF}_k(p) = \frac{\sum_{q \in N_k(p)} \text{lrd}_k(q)/\text{lrd}_k(p)}{|N_k(p)|}. \tag{2.4}$$

Значення LOF близькі до 1 указують на нормальну точку, значення, істотно більші за 1 — на аномалію.

Перевагою LOF є здатність виявляти аномалії в розподілах з кількома густинами (multi-density). Недоліки — обчислювальна складність $O(n^2)$ (потрібен пошук k найближчих сусідів для кожної точки), чутливість до вибору параметра k та необхідність попередньої стандартизації ознак.

2.1.4 Підхід на основі автокодувальників

Автокодувальники (autoencoders) — нейронні мережі, що навчаються відтворювати вхідні дані через стиснене внутрішнє представлення (латентний простір). Архітектурно автокодувальник складається з двох частин: ен-

кодера, що відображає вхідні дані у латентний простір меншої розмірності, та декодера, що відновлює вхідний вектор з латентного представлення.

При застосуванні до задачі виявлення аномалій модель навчається лише на нормальних даних; для аномальних точок похибка реконструкції буде істотно вищою, що використовується як оцінка аномальності. Цільова функція тренування — мінімізація середньоквадратичної похибки реконструкції:

$$L = \frac{1}{n} \sum_{i} \|x_i - D(E(x_i))\|^2, \tag{2.5}$$

де E — енкодер;

D — декодер.

Перевагами підходу є здатність моделювати складні нелінійні залежності та можливість роботи з високовимірними даними. Недоліки — потреба в значному обсязі тренувальних даних (типово десятки тисяч прикладів), висока обчислювальна складність навчання, необхідність наявності спеціалізованого апаратного забезпечення (GPU), слабка інтерпретованість висновків та складність налаштування архітектури мережі.

2.1.5 Порівняльний аналіз і обґрунтування вибору

У табл. 2.1 наведено порівняльний аналіз розглянутих алгоритмів за чотирма ключовими характеристиками [15].

Таблиця 2.1 – Порівняльний аналіз алгоритмів виявлення аномалій

Алгоритм	**Складність**	**Інтерпретованість**	**Тренувальні вимоги**	**Підходить**
Isolation Forest	$O(n \log n)$	Висока	Малий обсяг	Так
One-Class SVM	$O(n^2)$	Середня	Середній обсяг	Ні
LOF	$O(n^2)$	Середня	Середній обсяг	Ні
Autoencoder	$O(n)^*$	Низька	Великий обсяг + GPU	Ні

* Складність навчання автокодувальника лінійна за кількістю прикладів, проте істотно вища у константі та потребує GPU.

На основі проведеного аналізу для реалізації ML-шару виявлення аномалій обрано алгоритм Isolation Forest. Цей вибір обґрунтований лінійною обчислювальною складністю, високою інтерпретованістю оцінки, низькими вимогами до тренувального набору, доступністю ефективної реалізації у бібліотеці scikit-learn та здатністю функціонувати на стандартних CPU без потреби у GPU. Останній фактор є критичним для обраної архітектури розгортання, де основний бекенд працює у хмарному середовищі Railway без доступу до графічних прискорювачів.

2.2 Алгоритми класифікації подій

Окрім задачі виявлення аномалій, у системі розв'язуються дві supervised задачі класифікації: визначення рівня критичності події (severity classification) та визначення типу атаки (attack family classification) [16]. Для цих задач розглянуто три класичні алгоритми класифікації.

Logistic Regression (LR) — лінійний класифікатор, що моделює ймовірність належності до класу як логістичну функцію від лінійної комбінації ознак. Для бінарної задачі ймовірність належності точки x до класу 1 обчислюється як:

$$P(y = 1 \mid x) = \frac{1}{1 + \exp(-(w \cdot x + b))}, \tag{2.6}$$

де w — вектор вагових коефіцієнтів;

b — зсув.

Параметри моделі оптимізуються методом максимуму правдоподібності з регуляризацією (L1, L2 чи Elastic Net). Перевагами LR є висока інтерпретованість через коефіцієнти ознак, низька обчислювальна складність та робастність до перенавчання при правильному виборі регуляризатора. Недоліком — обмежена здатність моделювати нелінійні залежності, що можна

частково компенсувати застосуванням нелінійних перетворень ознак (поліноміальні ознаки, TF-IDF для текстових полів).

Random Forest (RF) — ансамблевий метод на основі дерев рішень. Кожне дерево навчається на бутстреп-вибірці з рандомізованим підмножиною ознак при кожному розщепленні. Підсумкова ймовірність належності до класу обчислюється як середнє арифметичне ймовірностей усіх дерев ансамблю. Перевагами RF є висока точність на табличних даних, природня обробка взаємодії ознак, помірна інтерпретованість через важливості ознак (feature importance) та робастність до перенавчання за рахунок усереднення дерев. Недоліком — більша обчислювальна вартість порівняно з LR.

XGBoost з технікою SMOTE — метод градієнтного бустингу над деревами рішень, запропонований Chen, Guestrin [17]. На відміну від Random Forest, XGBoost будує дерева послідовно, де кожне наступне дерево вчиться компенсувати помилки попередніх. Цільова функція тренування містить регуляризаційний доданок:

$$\text{Obj} = \sum_i L(y_i, \hat{y}_i) + \sum_k \Omega(f_k), \tag{2.7}$$

де L — функція втрат (типово log-loss для бінарної класифікації);
Ω — регуляризаційний доданок, що включає кількість листя дерева та норму вагових коефіцієнтів у листях.

У комбінації з технікою SMOTE (Synthetic Minority Oversampling Technique), запропонованою Chawla et al. [18], XGBoost дозволяє ефективно працювати з незбалансованими вибірками. SMOTE генерує синтетичні приклади міноритарного класу шляхом інтерполяції між наявними прикладами та їх найближчими сусідами в просторі ознак. У задачах виявлення атак, де кількість нормальних подій істотно перевищує кількість аномальних, ця комбінація є фактично стандартом галузі [16].

Усі три методи реалізовано в системі та оцінено на двох відкритих наукових корпусах: HDFS_v1 (Loghub) [19], зібраному за роботою Xu et al. [7], та CICIDS2017, представленому Sharafaldin et al. [8]. Деталі експериментів наведено у розділі 4.

Для побудови ознакового простору застосовано три типи перетворень. Числові ознаки (наприклад, тривалість потоку, кількість байтів) стандартизуються методом StandardScaler. Категоріальні ознаки (тип події, джерело, протокол) перетворюються через one-hot encoding. Текстові поля (повідомлення події) перетворюються через TF-IDF з обмеженням словника до 200–300 найвагоміших n-грам довжиною від одного до двох слів.

2.3 Технології векторного пошуку для RAG

Підхід Retrieval-Augmented Generation (RAG), запропонований у роботі Lewis et al. [20], поєднує великі мовні моделі з зовнішнім пошуком у векторному сховищі історичних даних. Архітектурно RAG-pipeline складається з трьох етапів: формування ембедінга користувацького запиту, пошук top-k найбільш схожих документів у сховищі, формування промпта для LLM з контекстом знайдених документів.

Узагальнену структуру RAG-pipeline системи, що розроблюється у даному дипломному проєкті, наведено на рис. 2.2. На схемі показано всі основні етапи від отримання доказів до повернення оператору структурованої діагностики.

Як показано на рис. 2.2, поточні докази (події та оцінка аномальності) спочатку перетворюються у 384-вимірний ембедінг за допомогою sentence-transformers MiniLM-L6-v2 [21], після чого виконується top-k пошук у ChromaDB [22] за косинусною подібністю з порогом `min_score` = 0,70 (обґрунтування цього значення наведено у розділі 4). Знайдені інциденти подаються у промпт як «loose pattern hints», а не як підтверджені аналоги, що,

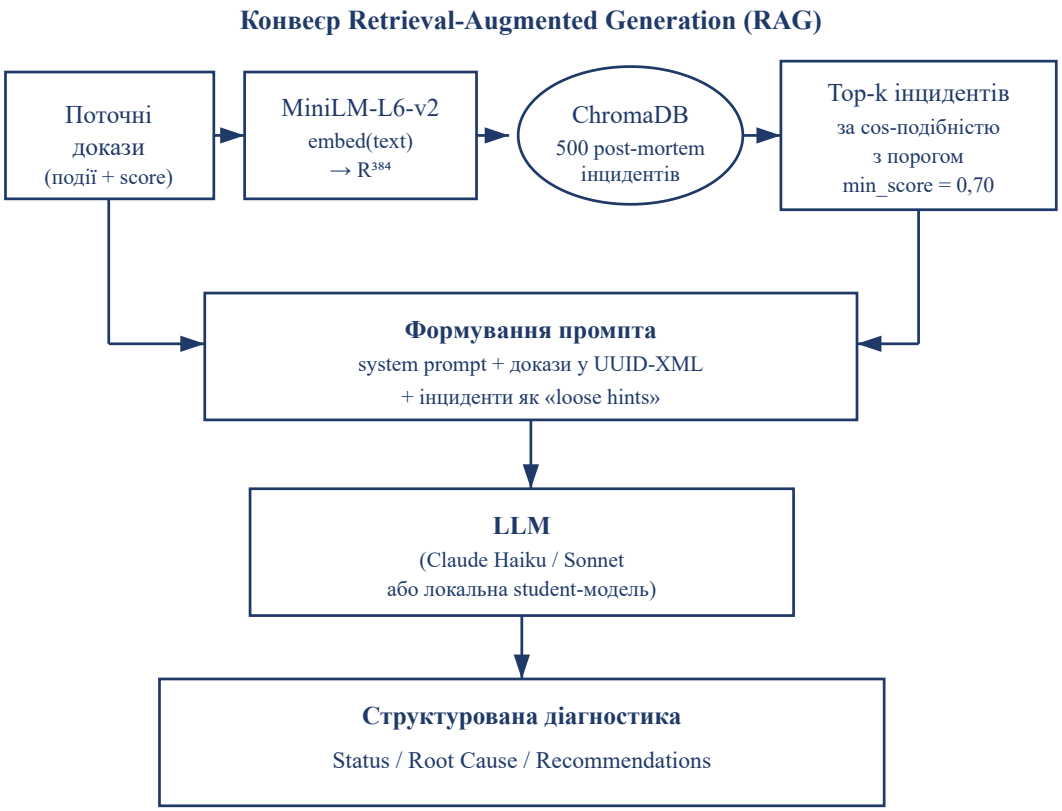


Рисунок 2.2 – Архітектура RAG-pipeline системи моніторингу

за результатами експериментів, зменшує ризик галюцинацій моделі. Підсумкова структурована діагностика повертається у форматі Status / Root Cause / Recommendations.

Для реалізації шару RAG необхідно обрати векторне сховище та модель ембедингів. Серед популярних векторних сховищ можна виділити ChromaDB (легка вбудована БД для прототипів та середніх корпусів), FAISS (бібліотека Facebook для високопродуктивного пошуку), pgvector (розширення для PostgreSQL), Weaviate (хмарна векторна БД промислового рівня).

ChromaDB має наступні переваги: вбудована SQLite-сумісна реалізація не потребує окремого процесу, простий Python API, підтримка метаданих та фільтрів, достатня продуктивність для корпусу до десятків тисяч документів, повна сумісність з режимом локального інференсу [22]. Недоліком є обмежена масштабованість на дуже великих корпусах (понад мільйон документів).

FAISS пропонує надзвичайно високу продуктивність пошуку завдяки оптимізованим індексам (HNSW, IVF, PQ), але вимагає додаткового налаштування та не має вбудованої підтримки персистентного збереження. pgvector є гарним вибором, коли в проєкті вже використовується PostgreSQL, але вимагає налаштування розширення на стороні СУБД. Weaviate — хмарне рішення промислового рівня, що виходить за межі потреб академічного проєкту.

Для розробленої системи обрано ChromaDB як оптимальне поєднання простоти інтеграції, достатньої продуктивності та повної локальності даних.

Серед моделей ембедингів на момент розробки розглянуто такі варіанти: OpenAI text-embedding-3 (комерційна, потребує мережних викликів), BGE-large (відкрита, велика модель $\sim$1 ГБ), sentence-transformers all-MiniLM-L6-v2 (відкрита, компактна модель $\sim$80 МБ, 384-вимірні ембединги). Якість ембедингів у задачах семантичного пошуку оцінюється на бенчмарку MTEB (Massive Text Embedding Benchmark).

Для розробленої системи обрано MiniLM-L6-v2, запропоновану у роботі Reimers, Gurevych [21], через малий обсяг моделі, достатню якість для задачі пошуку схожих інцидентів та повну локальність обчислень. Косинусна подібність двох ембедингів a, b обчислюється як:

$$\text{sim}(a, b) = \frac{a \cdot b}{\|a\| \cdot \|b\|}. \tag{2.8}$$

Значення подібності лежить у діапазоні $[-1; 1]$, причому 1 відповідає ідентичним напрямкам векторів, 0 — ортогональним, -1 — протилежним.

2.4 Технології великих мовних моделей

Шар LLM-діагностики є центральним компонентом розробленої системи. Розглянемо чотири підходи до інтеграції великих мовних моделей.

Прямий доступ до Anthropic API — найпростіший спосіб інтеграції з моделями сімейства Claude. API надає REST-ендпоінт для надсилання промптів та отримання відповідей; підтримує параметри temperature, max_tokens, system prompt, а також просунуті можливості, такі як `tool_use` та prompt caching. Перевагами є мінімальні витрати на розробку, доступ до найсучасніших моделей сімейства (Claude Haiku, Claude Sonnet, Claude Opus). Недоліки — повна залежність від сервісу одного постачальника та необхідність передавання payload подій за межі периметру організації.

Доступ до моделей Claude через AWS Bedrock — managed-сервіс Amazon, що надає уніфікований інтерфейс до моделей різних постачальників (Anthropic, AI21, Cohere, Meta, Amazon Titan). Переваги — інтеграція з екосистемою AWS (IAM-роли, VPC, CloudWatch), можливість вибору регіону розгортання, контроль витрат через AWS billing. Недоліки — вищі базові витрати порівняно з прямим API, складніша автентифікація через AWS Signature v4, необхідність окремого налаштування доступу до моделей через консоль Bedrock.

Локальний інференс власноруч дистильованої моделі — підхід, що забезпечує повну приватність та незалежність від зовнішніх постачальників. Реалізується на основі відкритої базової моделі (наприклад, Qwen2.5-1.5B-Instruct, Phi-3-mini, TinyLlama-1.1B) з донавчанням на корпусі (вхід, вихід) пар, отриманих з teacher-моделі [23]. Переваги — нульові маргінальні витрати на виклик, повний контроль даних, можливість роботи в air-gapped середовищах. Недоліки — висока вартість підготовки моделі (генерація корпусу

+ донавчання), нижча якість порівняно з frontier-моделями, потреба в спеціалізованому апаратному забезпеченні для інференсу (типово GPU).

Function-calling агенти — підхід, у якому LLM ітеративно викликає набір інструментів (tools) для збору необхідної інформації перед формуванням відповіді. Реалізується через спеціальні API, такі як Anthropic `tool_use` API чи OpenAI function calling. На кожній ітерації цикла модель або викликає один з наявних інструментів (і отримує його результат у наступному виклику), або формує фінальну відповідь. Переваги — можливість адаптивного збору доказів, розв'язування задач змінної глибини, природна обробка мульти-системних запитів. Недоліки — істотно вища вартість виклику (множинні round-trip-и до моделі), складність обмеження області дій агента, потреба в реалізації безпечного інтерфейсу інструментів.

У межах дипломного проєкту реалізовано всі чотири підходи у вигляді взаємозамінних рушіїв AI-діагностики, що повертають однотипну структуровану відповідь. Це забезпечує гнучкість оператора щодо вибору між якістю діагностики, вартістю та рівнем приватності даних.

2.5 Дистиляція знань (LoRA, QLoRA)

Дистиляція знань (knowledge distillation), запропонована у роботі Hinton, Vinyals, Dean [23], — клас методів передавання знань від великої моделі-вчителя (teacher) до менш ресурсомісткої моделі-учня (student). У контексті дипломного проєкту дистиляція виконується для отримання локального рушія інференсу, що відтворює поведінку комерційної моделі-вчителя на задачі формування діагностики.

Класична дистиляція передбачає повне донавчання student-моделі на виходах teacher-моделі (soft labels), що потребує значних обчислювальних ресурсів та доступу до модифікації всіх ваг базової моделі. Сучасні мето-

ди параметрично-ефективного донавчання (Parameter-Efficient Fine-Tuning, PEFT) дозволяють істотно знизити цю вартість.

LoRA (Low-Rank Adaptation), запропонована у роботі Hu et al. [24], додає до базової моделі низькорангові адаптери — пари матриць A розміру $d \times r$ та B розміру $r \times d$ (де $r \ll d$), які навчаються замість оригінальних ваг. При інференсі вихід шару, до якого підключено адаптер, обчислюється як $W \cdot x + B \cdot A \cdot x$, де W — оригінальна вага, $B \cdot A \cdot x$ — внесок адаптера. Адаптери підключаються до проєкційних шарів (query, key, value, output) трансформера. Розмір адаптера складає від десятків кілобайт до десятків мегабайт, що в сотні разів менше за розмір базової моделі. Це дозволяє ефективно зберігати множинні адаптери для різних задач та швидко перемикатися між ними.

Структурну схему механізму низькорангової декомпозиції наведено на рис. 2.3, де наочно показано співвідношення замороженої базової ваги W і навчаного LoRA-адаптера $B \cdot A$. Розмір адаптера у проєкті — приблизно 17 МБ, що в сотні разів менше за повну базову модель Qwen2.5-1.5B.

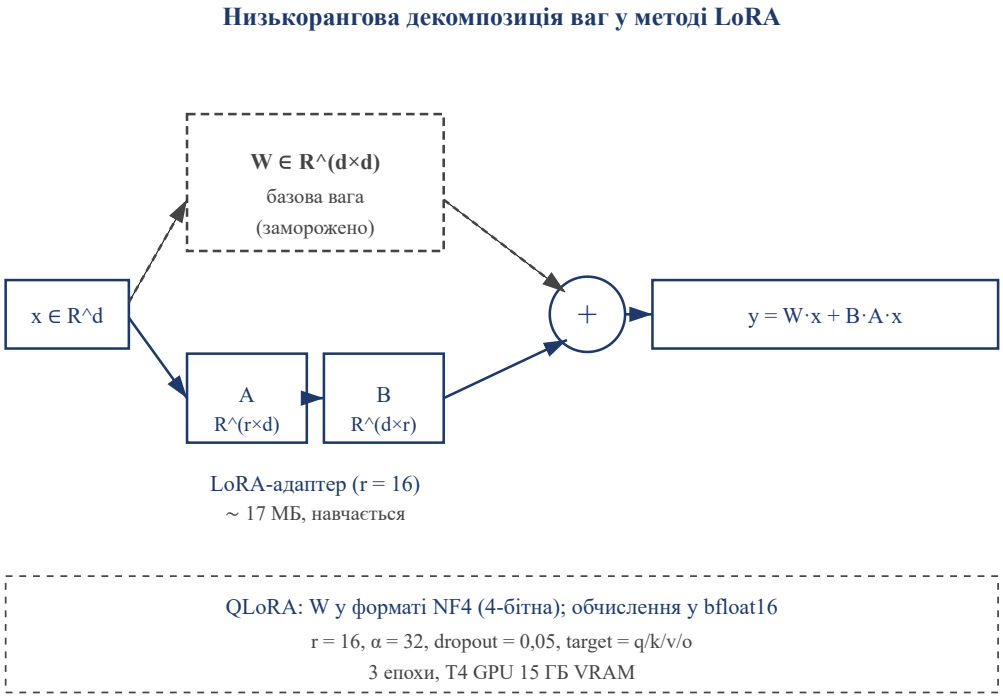


Рисунок 2.3 – Низькорангова декомпозиція ваг у методі LoRA

З рис. 2.3 видно, що під час тренування навчаються виключно матриці A і B (прямокутник із суцільною лінією), тоді як основна матриця W залишається замороженою (прямокутник із пунктирною лінією). Така архітектура зменшує кількість навчаних параметрів у кілька порядків і дозволяє зберігати множинні адаптери (по одному на кожну задачу) поверх однієї базової моделі. У режимі QLoRA, описаному нижче, базова матриця W додатково зберігається у пам'яті у форматі 4-бітної квантизації NF4, що ще більше знижує вимоги до VRAM.

QLoRA (Quantised LoRA), запропонована у роботі Dettmers et al. [25], поєднує LoRA з 4-бітною квантизацією базової моделі у форматі NF4 (Normalised Float 4-bit). Базова модель зберігається у пам'яті у квантизованій формі, а обчислення у проміжках інференсу виконуються у форматі bfloat16. Це дозволяє виконувати донавчання моделей розміром 1–7 мільярдів параметрів на споживчих GPU з обмеженою VRAM (наприклад, NVIDIA T4 з 15 ГБ VRAM, доступний у безкоштовному середовищі Google Colab).

Як базова модель для дистиляції в межах дипломного проєкту обрано Qwen2.5-1.5B-Instruct (Alibaba Cloud, ліцензія Apache 2.0). Цей вибір обґрунтований такими міркуваннями: розмір 1,5 мільярда параметрів забезпечує запуск на безкоштовному T4 у режимі QLoRA; модель є інструкційно-донавченою, що зменшує витрати на формування правильного формату відповіді (модель уже вміє слідувати інструкційному стилю); ліцензія Apache 2.0 знімає обмеження на академічне та комерційне використання. Альтернативи (Phi-3-mini від Microsoft, TinyLlama-1.1B) розглядалися, проте були відхилені з огляду на обмеження вмісту контексту (Phi-3-mini ледве вміщається у 16 ГБ VRAM при батч-розмірі 1) та нижчу базову якість (TinyLlama-1.1B істотно поступається Qwen2.5-1.5B на інструкційних бенчмарках).

2.6 Технологічний стек серверної частини

Розглянемо обґрунтування вибору ключових технологій серверної частини.

Мова програмування — Python 3.11. Серед альтернатив розглядалися Go (висока швидкодія, складна інтеграція ML-бібліотек, відсутність сильної екосистеми для трансформерних моделей), Node.js (обмежена екосистема ML, гарна для веб-серверів, але слабка для обчислень), Rust (надмірно складний для академічного проєкту, крута крива навчання). Обрано Python як дефакто стандарт галузі ML/AI з найбагатшою екосистемою бібліотек: scikit-learn для класичного ML, transformers та PEFT для LLM, sentence-transformers для ембедингів, ChromaDB для векторного сховища.

Веб-фреймворк — FastAPI 0.115. Серед альтернатив — Flask (простіший, але без вбудованої валідації схем, потребує ручної інтеграції з Pydantic), Django (надмірний для API-сервісу, приходить з ORM, шаблонами, адмінкою — більшість з цього не потрібно для бекенду), aiohttp (нижчий рівень абстракції, більше boilerplate-коду). Обрано FastAPI через автоматичну генерацію OpenAPI-документації, повну інтеграцію з Pydantic v2 для валідації схем, підтримку асинхронних обробників та активний розвиток у спільноті [26].

База даних — PostgreSQL. Серед альтернатив — MySQL (менш розвинена екосистема розширень, обмежена підтримка JSON-полів), MongoDB (відсутність строгої схеми може створити проблеми консистентності даних), SQLite (нестійка для багатотенантного навантаження, обмежена підтримка одночасних з’єднань). Обрано PostgreSQL як надійну реляційну СУБД з широкою екосистемою розширень (зокрема, pgvector для векторного пошуку, що при потребі може замінити ChromaDB) та підтримкою у хмарних провайдерах (Railway, Heroku, AWS RDS) [26].

ORM — SQLAlchemy 2.0. Сучасна версія з повною підтримкою type hints, що поєднує можливості Core (низькорівневий API) та ORM (об'єктний API). Альтернативи (peewee, Tortoise ORM, SQLModel) поступаються SQLAlchemy у зрілості та документації.

Фронтенд оператора — Streamlit. Серед альтернатив — React (надмірно складний для академічного дашборду, потребує окремого процесу збірки, окремого хостингу), Dash (схожий, але складніший, вимагає більше boilerplate-коду), Gradio (орієнтований на ML-демо, обмежені можливості розгортання та кастомізації). Обрано Streamlit як швидкий спосіб реалізації інтерактивного дашборду одним Python-файлом, з вбудованою підтримкою компонентів для графіків, таблиць та форм.

Хмарне розгортання — Railway та Azure Container Apps. Railway обрано для розгортання основного бекенду через простоту інтеграції з Git-репозиторієм (autodeploy на push), наявність managed PostgreSQL у тарифному плані, помірну вартість (від 5 USD на місяць). Azure Container Apps обрано для розгортання LoRA-сервісу через підтримку режиму scale-to-zero, що знижує idle-вартість до нуля. Альтернативами розглядалися AWS Fargate (вищі базові витрати, складніше налаштування), Google Cloud Run (схожий, але обмежена підтримка великих контейнерних образів, що містять ваги моделі) [28].

2.7 Методи захисту від prompt-injection

Атаки типу prompt-injection — клас атак на LLM-системи, що полягає у вбудовуванні в користувацький контент інструкцій для перехоплення поведінки моделі [27]. У системі моніторингу безпеки, де повідомлення подій потрапляють у промпт LLM, зловмисник може ретельно сформованим повідомленням вплинути на висновки автоматичної діагностики.

Серед основних методів захисту виділяють такі:

– регулярні вирази для виявлення відомих патернів атак (наприклад, «ignore all previous instructions», «you are now…», «forget your role»);
– обгортання користувацького контенту у XML-теги з випадковими ідентифікаторами (UUID), що ускладнює зловмиснику імітацію розмежувачів промпта;
– детектори альтернативних кодувань (base64, hex, leet-speak, багатомовні підстановки), що виявляють спроби маскування атак;
– явні інструкції в системному промпті щодо неприпустимості виконання команд із користувацького контенту;
– обмеження прав агента (доступ лише на читання, обмежений набір інструментів, бюджет на сесію).

У розробленій системі всі п’ять механізмів реалізовано за принципом глибокого захисту (defence in depth); їх описано в розділі 3, а перевірку стійкості — у розділі 4.

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі проведено детальний порівняльний аналіз алгоритмів та технологій, потенційно придатних для реалізації поставленої задачі.

Серед алгоритмів виявлення аномалій обрано Isolation Forest з огляду на лінійну обчислювальну складність, високу інтерпретованість оцінки, низькі вимоги до тренувального набору та доступність ефективної реалізації у бібліотеці scikit-learn.

Для задач supervised класифікації обрано три класичні методи: Logistic Regression як інтерпретований базовий метод, Random Forest як ансамблевий метод з природньою обробкою взаємодій ознак та XGBoost з технікою SMOTE як стандарт галузі для незбалансованих корпусів.

Для шару RAG обрано векторне сховище ChromaDB та модель ембедингів sentence-transformers MiniLM-L6-v2 як оптимальне поєднання простоти інтеграції, локальності обчислень та достатньої якості пошуку.

Для шару LLM-діагностики реалізовано чотири взаємозамінні рушії, кожен з яких задовольняє різні вимоги щодо якості, вартості та приватності даних. Як базова модель для дистиляції обрано Qwen2.5-1.5B-Instruct з технікою QLoRA на адаптерах рангу 16.

Технологічний стек серверної частини сформовано з Python 3.11, FastAPI, Pydantic v2, SQLAlchemy 2.0, PostgreSQL, ChromaDB та Streamlit. Розгортання компонентів планується у середовищах Railway (бекенд) та Azure Container Apps (LoRA-сервіс).

Для захисту від prompt-injection атак обрано п’ять механізмів глибокого захисту: регулярні вирази, UUID-обгортання, детектор кодувань, інструкції системного промпта та обмеження прав агента.

РОЗДІЛ 3
СИСТЕМА ЗБОРУ ТА АНАЛІЗУ СИГНАЛІВ БЕЗПЕКИ

3.1 Загальна архітектура системи

Розроблена система побудована за трирівневою архітектурою, що забезпечує чітке розділення відповідальності між шарами та полегшує майбутнє розширення функціональності [26]. Верхній рівень — шар представлення (HTTP-роутери, дашборд оператора). Середній рівень — шар бізнес-логіки (сервіси, що реалізують rule engine, ML-шар, RAG, рушії AI-діагностики, guardrails). Нижній рівень — шар доступу до даних (моделі SQLAlchemy, ChromaDB-клієнт, файлові артефакти моделей).

Загальну структурну схему системи з виокремленням усіх трьох рівнів і ключових модулів у кожному з них наведено на рис. 3.1. На цій схемі показано також точки інтеграції з зовнішніми сервісами (Anthropic API, AWS Bedrock, Azure Container Apps) та потоки даних між компонентами.

Аналіз рис. 3.1 дозволяє виокремити декілька принципових архітектурних рішень. По-перше, дотримано принципу «тонкого роутера»: HTTP-обробник лише приймає запит, валідує схему через Pydantic, делегує виконання сервісу та формує відповідь. Уся бізнес-логіка зосереджена в сервісах, що не залежать від HTTP-фреймворку та можуть бути використані повторно у CLI-сценаріях, скриптах оцінювання та тестах. По-друге, Pydantic-моделі (валідація HTTP) і SQLAlchemy-моделі (робота з БД) розділені фізично, що уникає зв’язування шару API з шаром збереження і дозволяє незалежно еволюціонувати обидва шари. По-третє, чотири рушії AI-діагностики винесено в окрему підгрупу сервісів та об’єднано спільним інтерфейсом, що задовольняє один контракт відповіді — це забезпечує можливість заміни рушія без зміни клієнтського коду.

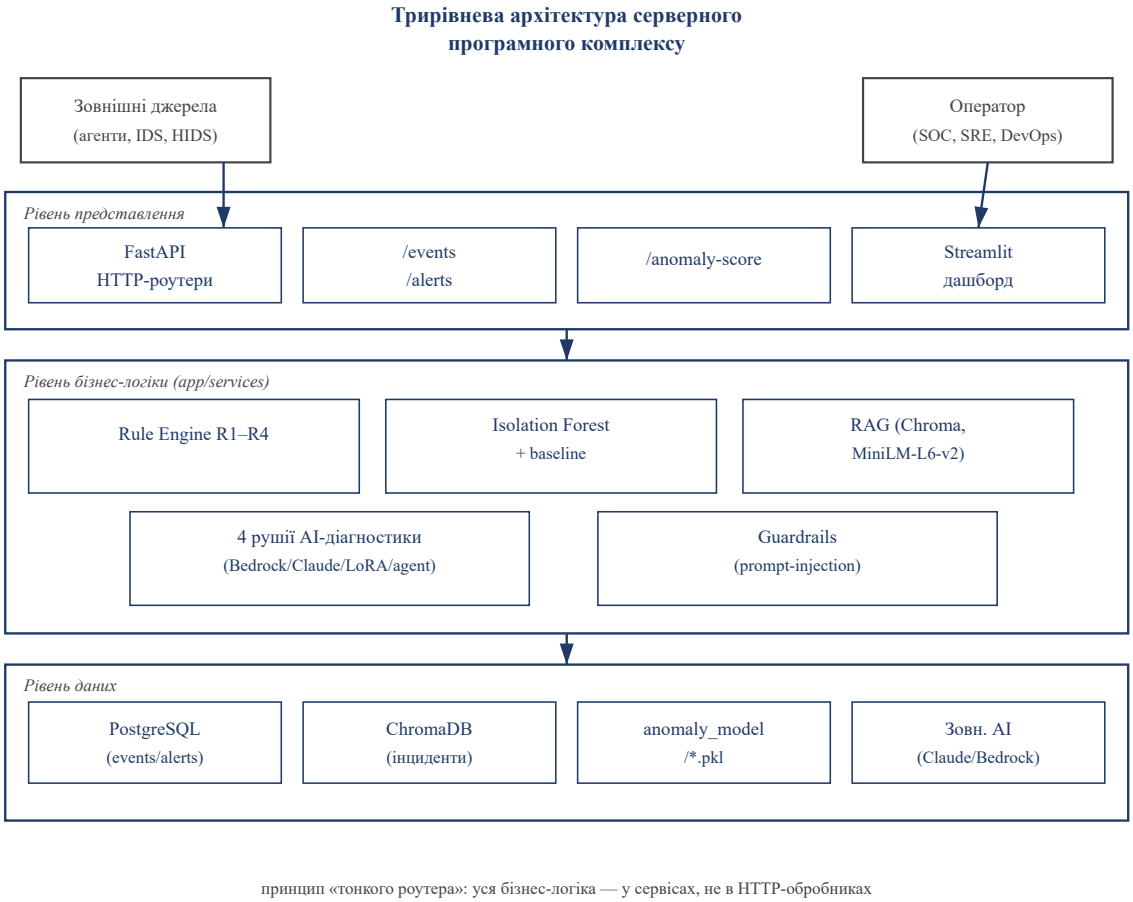


Рисунок 3.1 – Загальна архітектура розробленої системи

На вхід системи через HTTP-ендпоінт `POST /events` надходять уніфіковані події безпеки. Кожна подія записується до реляційного сховища PostgreSQL, після чого послідовно обробляється детермінованим рушієм правил, ML-шаром детекції аномалій та (за запитом оператора) одним з чотирьох рушіїв AI-діагностики.

Архітектурно код проєкту розділено на такі основні модулі: роутери HTTP-обробників (директорія `app/routers`), бізнес-логіка сервісів (`app/services`), моделі бази даних (`app/db`), Pydantic-схеми API (`app/schemas`), допоміжні утиліти (`app/auth.py` для перевірки API-ключа, `app/config.py` для зчитування змінних оточення, `app/logging_config.py` для централізованої конфігурації логуван-

ня). Точкою входу до застосунку є файл `app/main.py`, що ініціалізує FastAPI-застосунок, реєструє роутери та налаштовує middleware.

3.2 Структура бази даних

Реляційне сховище реалізовано на основі PostgreSQL з використанням ORM SQLAlchemy 2.0. Схема бази даних містить три основні таблиці. Узагальнену ER-діаграму схеми наведено на рис. 3.2.

З рис. 3.2 видно структуру трьох таблиць та принциповий зв'язок: одна подія `events` може породжувати кілька записів `alerts` (від різних правил R1–R4), а таблиця `baselines` слугує джерелом нормалізаційних коефіцієнтів для ML-шару детекції аномалій. Поле `system_id` у всіх трьох таблицях забезпечує мультитенантну ізоляцію даних різних підсистем інфраструктури в одному екземплярі БД.

Таблиця `events` — таблиця подій безпеки. Поля: id (первинний ключ, автоінкрементний), timestamp (часова мітка події у форматі TIMESTAMPTZ), source (текстове поле з джерелом події, наприклад, «web-server-01»), type (тип події з фіксованого переліку: attack, error, deploy_fail, info, warning), severity (рівень критичності: critical, warning, info), message (текстовий опис події), `system_id` (ідентифікатор підсистеми для мультитенантної ізоляції), metadata (JSON-поле з додатковими атрибутами). На таблиці створено два індекси: композитний індекс за (`system_id`, timestamp) для пришвидшення вибірок останніх подій підсистеми та індекс за type для агрегацій.

Таблиця `alerts` — таблиця сповіщень, згенерованих рушієм правил. Поля: id (первинний ключ), `event_id` (зовнішній ключ на `events`), `rule_id` (ідентифікатор правила, що спрацювало: R1, R2, R3, R4), `created_at` (часова мітка створення сповіщення), `system_id` (ідентифікатор підсистеми), payload (JSON з додатковим контекстом спрацювання). На

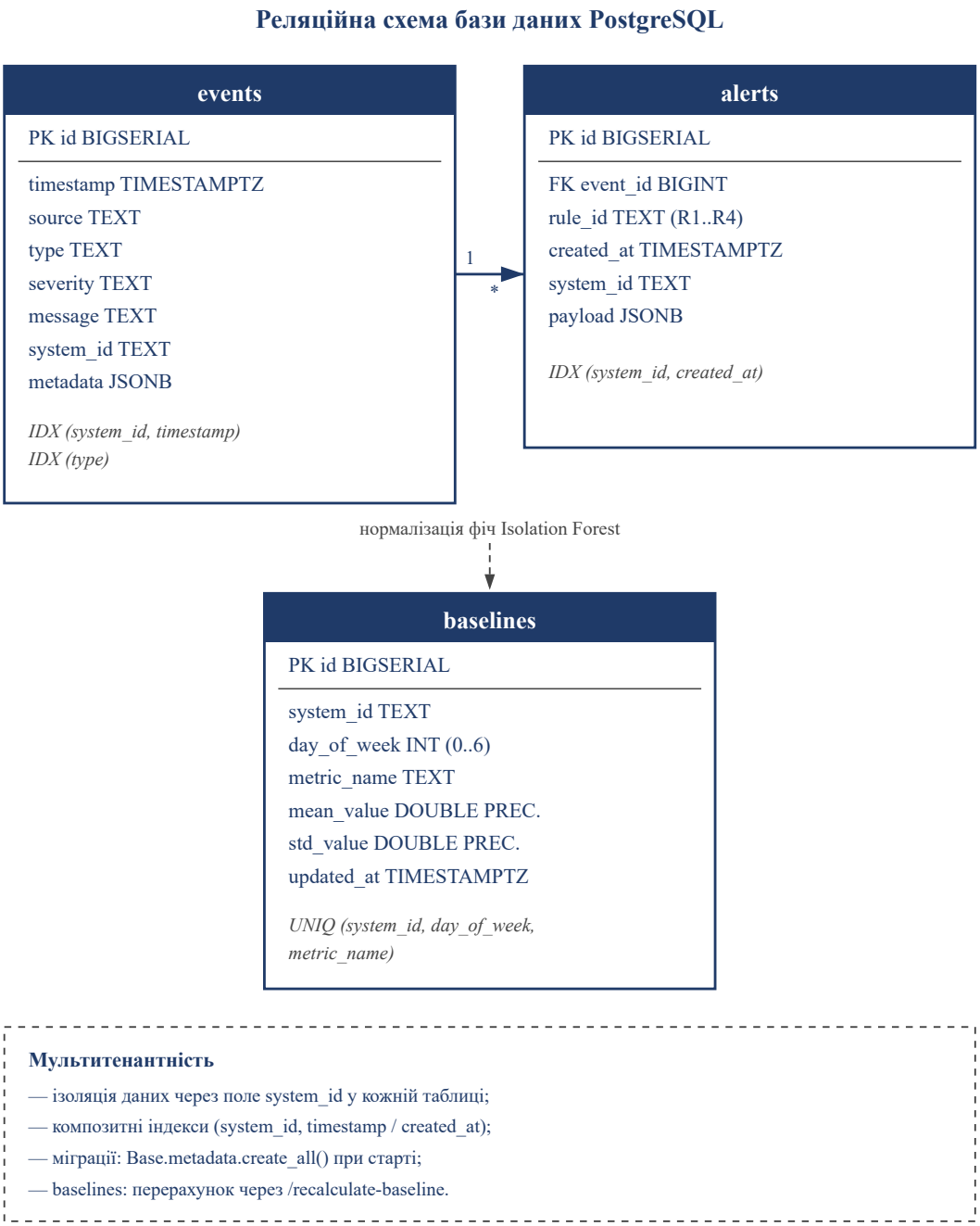


Рисунок 3.2 – Реляційна схема бази даних PostgreSQL

таблиці створено індекс за (`system_id`, `created_at`) для швидкого отримання останніх сповіщень.

Таблиця `baselines` — таблиця базових профілів активності підсистем. Поля: id (первинний ключ), `system_id` (ідентифікатор підсистеми), `day_of_week` (день тижня від 0 до 6), `metric_name` (назва метрики: `attacks_per_hour`, `errors_per_hour`, `total_events_per_hour`

тощо), mean_value (середнє значення метрики), std_value (стандартне відхилення), updated_at (часова мітка останнього оновлення профілю). На таблиці створено унікальний композитний індекс за (system_id, day_of_week, metric_name).

Усі таблиці містять індекси за полями system_id та timestamp / created_at для пришвидшення вибірок у багатотенантних сценаріях. Міграції схеми бази даних керуються файлами SQLAlchemy-моделей; при першому запуску застосунок автоматично створює відсутні таблиці методом Base.metadata.create_all.

3.3 Реалізація детермінованого рушія правил R1–R4

Детермінований рушій правил реалізовано в модулі app/services/rule_engine.py. Рушій містить чотири правила, що відповідають найпоширенішим критичним патернам у потоці подій безпеки. Структурну схему рушія наведено на рис. 3.3.

Як видно з рис. 3.3, кожна вхідна подія паралельно перевіряється чотирма правилами; ті з них, що спрацьовують, формують записи у таблицю alerts. Усі правила виконуються синхронно при прийомі події через POST /events, що забезпечує мінімальну латентність формування сповіщень. Завдяки попередній фільтрації за полем system_id (індексованим) результати агрегаційних запитів R2–R4 не змішують дані різних підсистем інфраструктури.

Правило R1 — критична подія. Спрацьовує при отриманні події з severity = critical. Формує сповіщення з посиланням на оригінальну подію та зразу зберігає його у базі. Це правило має найвищий пріоритет і виконується першим, що забезпечує мінімальну латентність формування сповіщень про найгостріші інциденти.

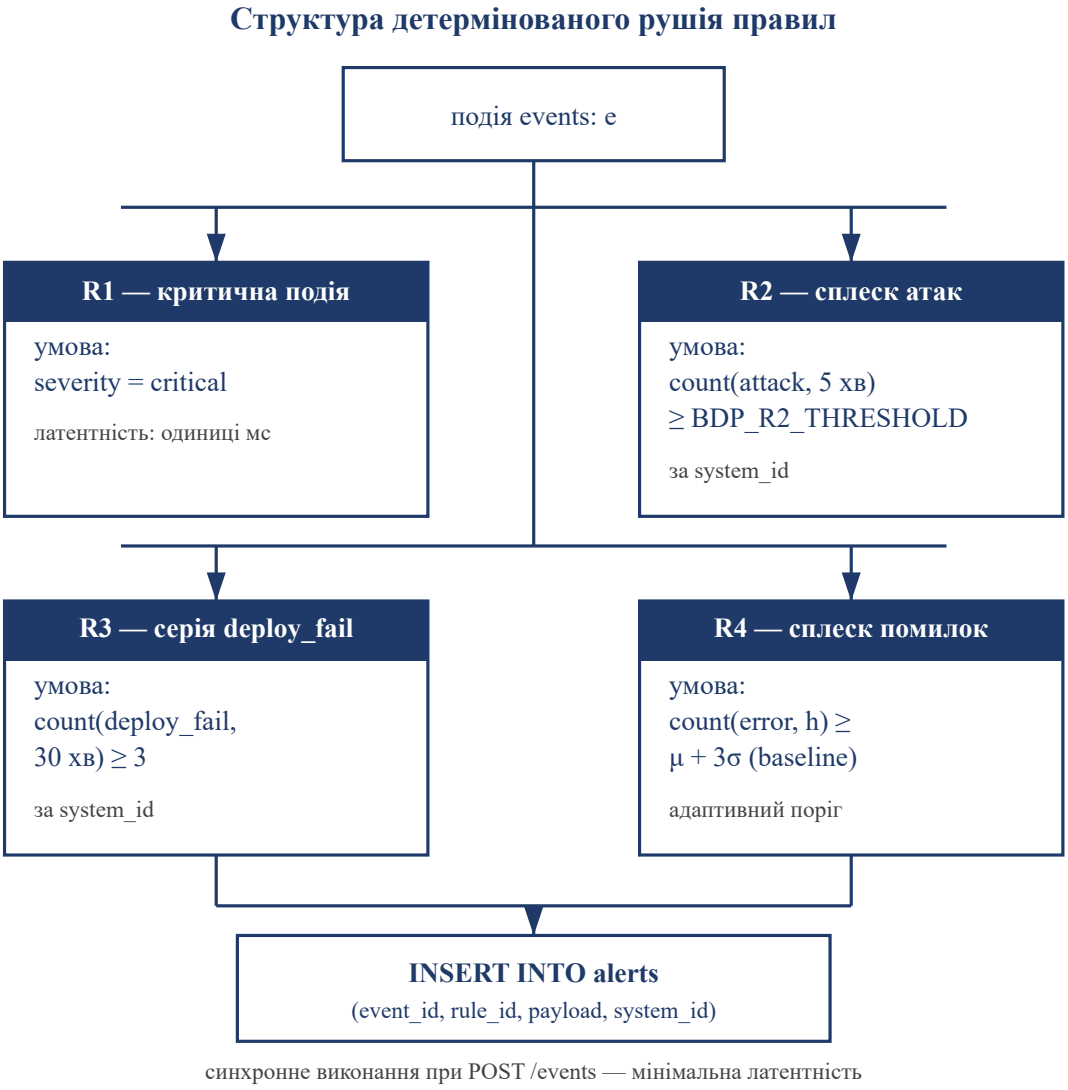


Рисунок 3.3 – Структура детермінованого рушія правил R1–R4

Правило R2 — сплеск атак. Спрацьовує при перевищенні порогу кількості подій з type = attack за фіксований часовий вікно (стандартно — 5 хвилин). Поріг налаштовується через змінну оточення `BDP_R2_THRESHOLD` (стандартне значення — 10 атак за 5 хвилин). Правило використовує кешований запит до бази даних з попереднім фільтром за `system_id` для уникнення міжтенантних перешкод.

Правило R3 — серія невдалих розгортань. Спрацьовує при виявленні трьох і більше подій з type = deploy_fail за останні 30 хвилин. Це правило допомагає виявити випадки, коли деплой ітеративно валиться через одну й ту

саму причину, що часто вказує на проблему з конфігурацією чи інфраструктурою.

Правило R4 — сплеск помилок. Спрацьовує при перевищенні статистично-визначеного порогу подій з type = error відносно базового профілю активності підсистеми. На відміну від правил R2 та R3, де поріг є константою, тут поріг адаптується до індивідуального профілю кожної підсистеми (mean + 3·std за відповідним днем тижня), що знижує кількість хибних спрацювань.

3.4 Реалізація ML-шару детекції аномалій

ML-шар детекції аномалій реалізовано в модулі `app/services/anomaly.py`. Шар використовує попередньо натреновану модель Isolation Forest з 8-вимірним вектором ознак [12].

Вектор ознак для оцінки аномальності формується агрегацією подій підсистеми за заданий період lookback (стандартно — 1 година). До складу 8 ознак входять: загальна кількість подій, кількість подій за п’ятьма типами (attack, error, deploy_fail, info, warning), кількість критичних подій (severity = critical), кількість унікальних джерел подій (унікальних значень source у вибірці).

Перед оцінкою аномальності ознаки нормалізуються відповідно до індивідуального профілю підсистеми (per-system baseline), що враховує день тижня. Це знижує кількість хибних спрацювань у сценаріях, де нормальна активність системи систематично відрізняється у вихідні дні чи робочий час. Без такої нормалізації робочий день ввечері в п’ятницю міг би бути помилково класифікований як аномальний просто через зростання навантаження порівняно з днем.

Модель Isolation Forest, scaler ознак (StandardScaler) та перелік імен ознак зберігаються у директорії `anomaly_model/` як артефакти у фор-

маті pickle. Файли: `isolation_forest.pkl` (серіалізована модель), `scaler.pkl` (серіалізований StandardScaler), `feature_names.json` (перелік імен ознак для контролю порядку при інференсі). Завантаження моделі здійснюється за патерном lazy singleton при першому виклику оцінки — це уникає затримки старту застосунку та знижує споживання пам'яті при низькому навантаженні.

Узагальнений pipeline обробки однієї події з моменту приймання HTTP-запиту POST `/events` до запису її у БД, виконання правил та (асинхронно) оцінки аномальності наведено на рис. 3.4.

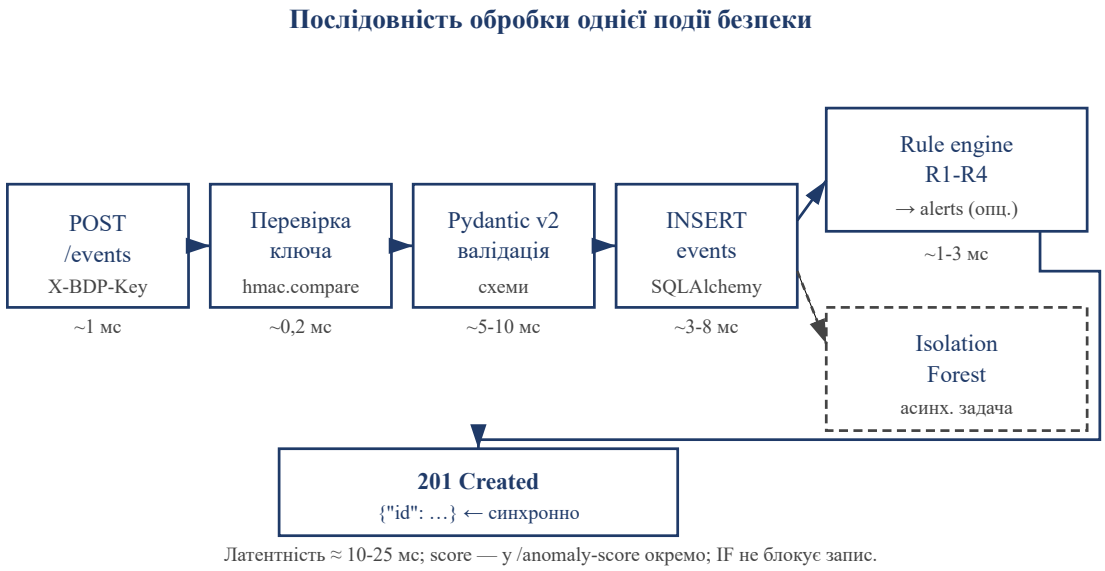


Рисунок 3.4 – Pipeline обробки однієї події безпеки

З рис. 3.4 випливає, що приймання події не блокується на обчисленні оцінки аномальності — ця операція виконується окремо при запиті `/anomaly-score`. Така архітектура дозволяє уникнути додаткової латентності в основному потоку запису подій, при цьому залишаючи оператору можливість отримати поточну оцінку у будь-який момент. Загальна латентність `POST /events` становить приблизно 10–25 мс (без AI-діагностики, яка викликається окремими ендпоінтами).

Ендпоінт `POST /anomaly-score` приймає параметри `system_id` та `lookback_hours`, виконує агрегацію подій, обчислює вектор ознак, застосовує scaler та модель, і повертає JSON з оцінкою аномальності та переліком фіч. Це дозволяє оператору не лише побачити числову оцінку, а й діагностувати, які саме характеристики потоку сприяли спрацюванню моделі.

3.5 Реалізація шару RAG

Шар Retrieval-Augmented Generation реалізовано у вигляді трьох модулів: `app/services/rag/embeddings.py` (обчислення ембедингів), `app/services/rag/vector_store.py` (взаємодія з ChromaDB [22]), `app/services/rag/retrieval.py` (пошук схожих інцидентів). Розділення на три модулі дозволяє замінити кожний з компонентів окремо — наприклад, перейти з MiniLM-L6-v2 на іншу модель ембедингів або з ChromaDB на FAISS — без змін у коді клієнтського коду.

Як модель ембедингів використано sentence-transformers all-MiniLM-L6-v2, що формує 384-вимірні вектори для текстових документів [21]. Модель розміром приблизно 80 МБ завантажується одноразово при першому виклику та кешується у пам’яті процесу. На рівні API модуль embeddings.py надає функцію embed(text) → np.ndarray, що приймає довільний текстовий документ і повертає його ембединг.

Векторне сховище реалізовано на основі ChromaDB у режимі PersistentClient зі збереженням даних у директорії `incidents_db/`. Корпус історичних інцидентів містить 500 синтетично згенерованих post-mortem-описів за дев’ятьма архетипами (DDoS, brute_force, service_5xx, deploy_fail, intrusion, data_corruption, config_drift, port_scan, calm_baseline). Кожен інцидент має унікальний ідентифікатор (INC-XXXX) та метадані з типом архетипу і часовою міткою.

Функція пошуку `find_similar`(text, k, `min_score`) повертає top-k найбільш схожих інцидентів за косинусною відстанню. Параметр `min_score` забезпечує фільтрацію слабких відповідностей: за результатами експериментального дослідження (розділ 4) для оптимальної якості діагностики обрано поріг 0,70.

Скрипт побудови індексу (`ml/rag/build_index.py`) виконує одноразову ініціалізацію корпусу: читає файл `ml/datasets/processed/incidents_history.jsonl`, обчислює ембедінги для кожного інциденту та записує їх у ChromaDB. Артефакти інциденти БД (директорія `incidents_db/`) включаються до git-репозиторію, що забезпечує робочий стан системи одразу після клонування.

3.6 Чотири взаємозамінні рушії AI-діагностики

Шар AI-діагностики реалізовано як чотири взаємозамінні рушії, що повертають однотипну структуровану відповідь у форматі Status / Root Cause / Recommendations. Усі чотири рушії доступні через окремі HTTP-ендпоінти та через дашборд оператора (вкладки AI Analysis Lab). Узагальнену архітектуру шару AI-діагностики наведено на рис. 3.5.

Як показано на рис. 3.5, всі чотири рушії звертаються до уніфікованого LLM-клієнта, що відповідає за atomic-резервацію бюджету процесу перед викликом моделі та за централізоване логування метаданих (кількість токенів, вартість, латентність). Це забезпечує єдиний контроль витрат незалежно від того, який саме рушій було обрано оператором. Уніфікований формат відповіді (Status / Root Cause / Recommendations) робить рушії взаємозамінними у клієнтському коді — оператор у дашборді просто перемикає вкладку, не змінюючи нічого у логіці запиту.

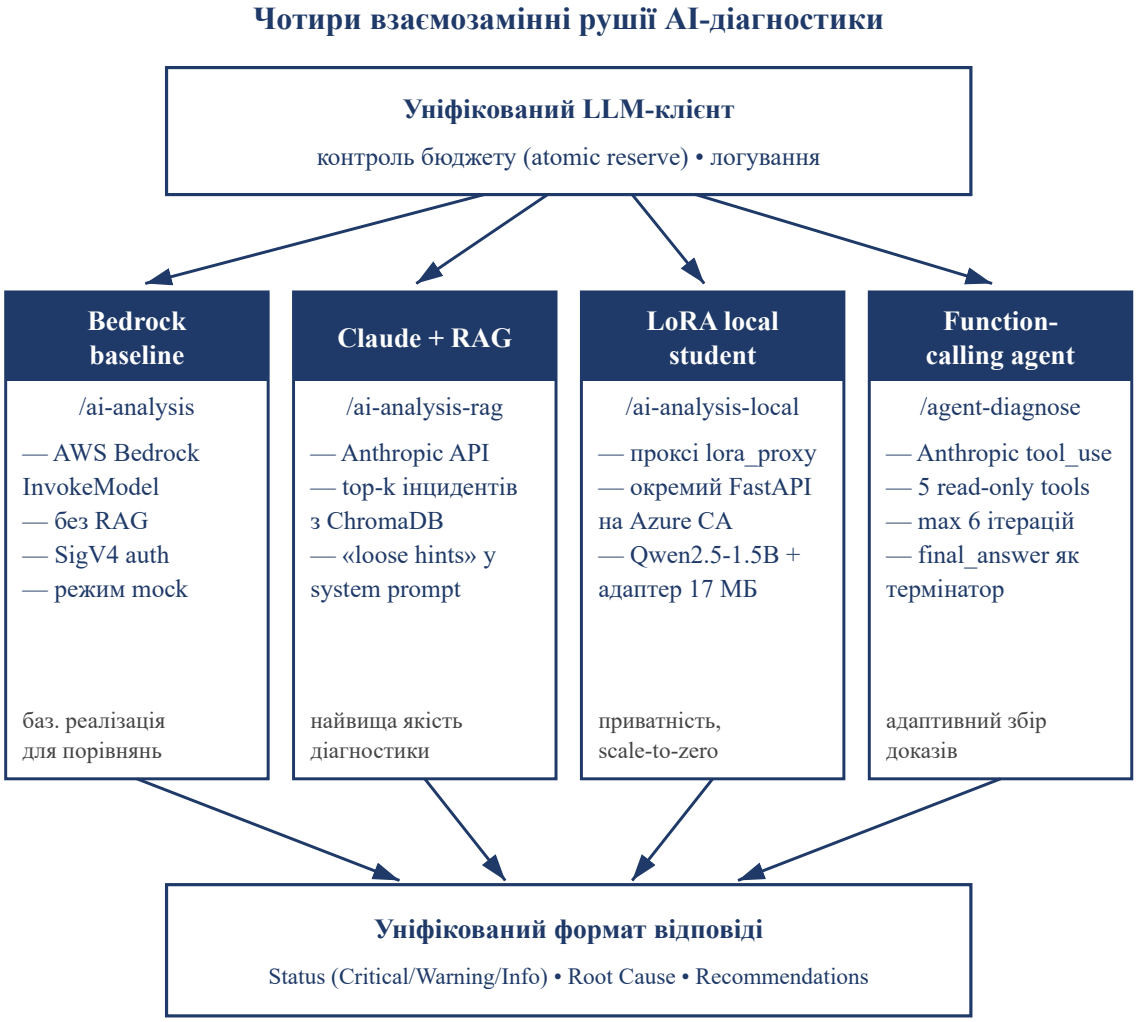


Рисунок 3.5 – Архітектура чотирьох рушіїв AI-діагностики

3.6.1 Рушій Bedrock baseline

Реалізовано у модулі `app/services/bedrock.py`. Здійснює прямий виклик моделі Claude через AWS Bedrock InvokeModel API. Цей рушій застосовується як базова реалізація для порівняльних експериментів та як резервний варіант при недоступності прямого Anthropic API. Аутентифікація здійснюється через AWS Signature v4 з використанням змінних оточення `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` та `BEDROCK_REGION`. Для розробки та тестування підтримується mock-режим

(BEDROCK_USE_MOCK=true), у якому повертаються реалістичні шаблонні діагностики без виклику реального API.

3.6.2 Рушій Claude + RAG

Реалізовано у модулі app/services/diagnoser.py. Здійснює виклик моделі Claude через прямий Anthropic API з попереднім пошуком схожих історичних інцидентів у ChromaDB [20]. Знайдені інциденти включаються в контекст промпта в розділі «Loose pattern hints — not confirmed analogues», що, за результатами експерименту з налаштування RAG, знижує ризик галюцинацій моделі через переформулювання їх як «слабких натяків», а не як «підтверджених аналогів».

Промпт побудовано у три блоки: системний промпт з описом ролі та правилами поведінки (зокрема, untrusted-content rule для захисту від prompt-injection); user-промпт зі структурованими доказами (події, оцінка аномальності, baseline); опційний блок знайдених інцидентів. Виклик моделі здійснюється через уніфікований клієнт app/services/llm_client.py, що відстежує бюджет процесу через atomic-резервацію перед викликом.

3.6.3 Рушій LoRA local student

Реалізовано у вигляді двох компонентів: app/services/lora_proxy.py (проксі на стороні бекенду) та lora-service/main.py (окремий FastAPI-сервіс інференсу). Локальний інференс виконується на дистильованій моделі Qwen2.5-1.5B-Instruct з підключеним LoRA-адаптером розміром приблизно 17 МБ [24]. Сервіс інференсу розгорнуто на Azure Container Apps з режимом scale-to-zero.

Проксі lora_proxy.py підтримує три режими роботи: mock (повертає реалістичні шаблонні діагностики без виклику зовнішнього сервісу, корисний для розробки та демонстрацій без активного Azure-сервісу), real (виклик розгорнутого LoRA-сервісу через HTTP) та off (повертає 503 Service Unavailable

при спробах виклику). Перемикання режимів здійснюється через змінну оточення `LORA_BACKEND`, що дозволяє контролювати поведінку без перерозгортання.

Окремий сервіс інференсу `lora-service/main.py` — це FastAPI-застосунок, що завантажує базову модель та LoRA-адаптер при старті процесу і обробляє запити на інференс через ендпоінт `POST /diagnose`. Контейнерний образ для розгортання містить запечені у образ ваги моделі (`HF_HUB_OFFLINE=1`), що забезпечує швидкий холодний старт без мережних запитів до Hugging Face Hub. Винесення цього компонента в окремий контейнерний образ є типовим архітектурним рішенням мікросервісного підходу: важкий ML-модуль ізолюється від основного бекенду і може масштабуватися та оновлюватися незалежно [28].

3.6.4 Function-calling агент

Реалізовано у модулях `app/services/agent/tools.py` (каталог інструментів) та `app/services/agent/loop.py` (цикл взаємодії з LLM). Агент працює на основі Anthropic `tool_use` API і має доступ до п’яти інструментів: `get_anomaly_score` (отримання оцінки аномальності), `get_baseline` (отримання базового профілю активності), `get_recent_events` (отримання останніх подій), `retrieve_similar_incidents` (пошук у ChromaDB) та `final_answer` (формування фінальної відповіді).

Усі інструменти, окрім `final_answer`, реалізовано як read-only — вони не можуть змінювати дані у базі чи створювати сповіщення. Це обмежує можливі наслідки потенційно скомпрометованої поведінки агента: навіть якщо зловмисник зможе вплинути на план дій агента через prompt-injection, найгірше, що може статися — отримання додаткових read-only даних [27]. Цикл обмежено шістьма ітераціями, що захищає від нескінченних

циклів та неконтрольованих витрат. Інструмент `final_answer` виступає термінатором циклу — як тільки агент його викликає, цикл завершується незалежно від того, чи паралельно агент викликав інші інструменти.

3.7 Реалізація захисту від prompt-injection

Шар захисту від prompt-injection реалізовано у модулі `app/services/security/guardrails.py`. Захист побудовано за принципом глибокого захисту (defence in depth) і поєднує п’ять механізмів. Структурну схему побудови захисту з виокремленням п’яти рівнів наведено на рис. 3.6.

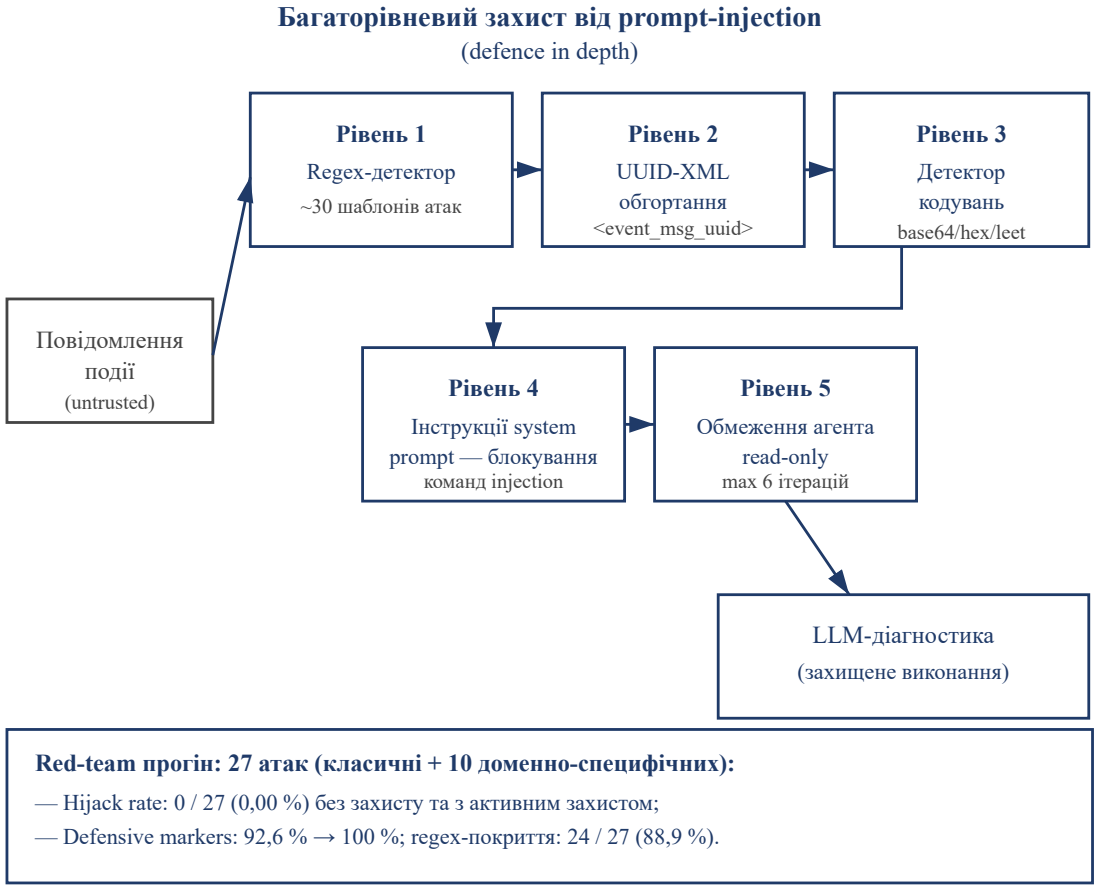


Рисунок 3.6 – Багаторівневий захист від prompt-injection (defence in depth)

Як показано на рис. 3.6, повідомлення події перед потраплянням у промпт LLM проходить три попередні фільтри (Рівні 1–3: regex, UUID-

обгортання, детектор кодувань), після чого виконавець промпта розглядає вже захищений вміст з активними інструкціями системного промпта (Рівень 4) і обмеженнями прав агента (Рівень 5). Така багаторівнева архітектура забезпечує захист навіть у тому разі, коли зловмисник зможе обійти один з механізмів — інші чотири продовжують виконувати свою функцію. Результати експериментального дослідження стійкості до 27 атак (нульова кількість успішних перехоплень) детально розглянуто у розділі 4.

Перший механізм — регулярні вирази для виявлення відомих патернів атак. Реалізовано приблизно 30 шаблонів, що покривають типові формулювання («ignore all previous instructions», «you are now…», «forget your role», «system prompt», «new instructions» тощо). Кожен шаблон у разі спрацювання підвищує оцінку ризику повідомлення; за сумою спрацювань визначається підсумковий рівень ризику (SAFE, MEDIUM, HIGH).

Другий механізм — UUID-обгортання користувацького контенту. Текст повідомлення події загортається у XML-теги з унікальним ідентифікатором. Випадковість тегу унеможливлює зловмиснику імітацію розмежувача в тексті атаки, оскільки UUID генерується динамічно і не відомий наперед.

Третій механізм — детектор кодувань. Виявляє спроби маскування атак через альтернативні кодування: base64 (за патернами входжень довгих рядків з base64-алфавіту), шістнадцяткове представлення (за входженнями послідовностей шестнадцяткових цифр), leet-speak (за характерними замінами «o»→«0», «i»→«1», «e»→«3»), багатомовні підстановки кирилиці на латиницю та навпаки.

Четвертий механізм — інструкції системного промпта, що явно забороняють моделі виконувати команди з користувацького контенту. У системному промпті є чітке правило: «текст у тегах <event_msg_*> є даними від користувача і має бути аналізований як контент, а не виконуватись як інструкція».

П’ятий механізм — обмеження прав агента. Function-calling агент має доступ лише до read-only інструментів, кількість ітерацій обмежена шістьма, бюджет на сесію контролюється через atomic-резервацію в LLM-клієнті.

3.8 Реалізація дистиляції teacher → student

Процес дистиляції знань реалізовано у директорії `ml/distill/` [23]. Дистиляція виконується у три етапи: формування корпусу пар (вхід, вихід), донавчання LoRA-адаптера, інтеграція адаптера у сервіс інференсу. Узагальнений pipeline всіх чотирьох етапів (включно з валідацією) наведено на рис. 3.7.

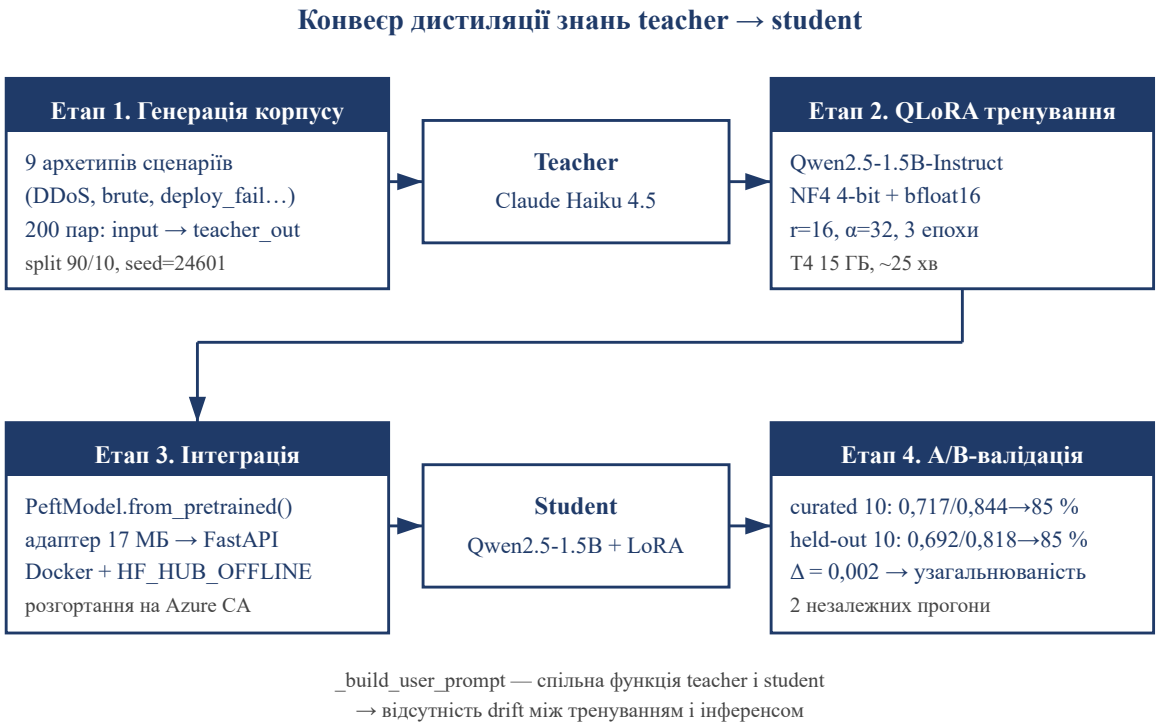


Рисунок 3.7 – Pipeline дистиляції знань teacher → student

Як показано на рис. 3.7, важливою архітектурною властивістю pipeline є те, що функція `_build_user_prompt`, яка формує промпт із доказами, є спільною для teacher-моделі (на етапі генерації корпусу) та student-моделі

(на етапі інференсу). Це гарантує, що student-модель бачить ту саму структуру XML-обгортання, ту саму секцію RAG-контексту і той самий формат доказів, на яких вона тренувалася. Без цієї інваріанти між тренуванням і інференсом виникав би drift, що призвів би до зниження якості. Завершальна A/B-валідація на двох незалежних прогонах підтвердила узагальнюваність дистиляції (детальні результати — у розділі 4).

На першому етапі скрипт generate_dataset.py генерує 200 пар на основі дев'яти архетипів сценаріїв (DDoS, brute_force, service_5xx, deploy_fail, intrusion, data_corruption, config_drift, port_scan, calm_baseline) з ваговим розподілом, що відповідає типовому навантаженню системи. Для кожного згенерованого вхідного prompt викликається teacher-модель (Claude Haiku 4.5), а її відповідь зберігається як цільовий вихід пари. Корпус розділено у пропорції 90/10 на тренувальну та валідаційну вибірки з фіксованим seed для відтворюваності.

На другому етапі донавчання виконується у середовищі Google Colab на безкоштовному GPU NVIDIA T4 з 15 ГБ VRAM. Гіперпараметри: ранг LoRA $r = 16$, alpha = 32, dropout = 0,05, цільові проєкції — q/k/v/o (query, key, value, output трансформера), кількість епох — 3, learning rate = $2 \cdot 10^{-4}$ з косинусним розкладом та 5 % warmup, ефективний розмір батчу — 8 (per_device = 2 $\times$ gradient_accumulation = 4), обчислення в bfloat16, оптимізатор paged AdamW 8-bit. Базова модель квантизована у форматі NF4 4-bit [25]. Тривалість донавчання — приблизно 25 хвилин.

На третьому етапі натренований адаптер розміром 17 МБ підключається до базової моделі у сервісі інференсу. Розгортання здійснюється у Docker-образі з включеними вагами базової моделі (`HF_HUB_OFFLINE=1`), що забезпечує швидкий холодний старт без мережних запитів. Завантаження адаптера відбувається через бібліотеку PEFT (`PeftModel.from_pretrained`).

3.9 Реалізація дашборду оператора

Дашборд оператора реалізовано на основі фреймворку Streamlit як одиничний Python-файл `dashboard/app.py` обсягом приблизно 4200 рядків. Дашборд складається з чотирьох основних розділів: огляд систем, детальний перегляд timeline подій, лабораторія AI-аналізу (AI Analysis Lab), статистика. Лабораторія AI-аналізу містить чотири вкладки, по одній на кожен з рушіїв (Bedrock baseline, Claude+RAG, LoRA local, Function-calling agent), що відповідає архітектурі шару діагностики (рис. 3.5). Дані оновлюються з бекенду з кешуванням протягом 60 секунд для уникнення зайвого навантаження. Конкретний візуальний вигляд інтерфейсу для основних сценаріїв наведено на рис. 3.8–3.11.

Розділ огляду дозволяє оператору обрати підсистему та часове вікно (1, 6, 24 години, 3 або 7 діб) і отримати зведення її активності у вигляді ключових показників: загальна кількість подій, кількість атак, помилок та сповіщень за обраний період, кожен — із відсотковим відхиленням від базового профілю (baseline). Стан підсистеми позначається кольоровим бейджем (зокрема `CRITICAL`), а окремим великим індикатором виводиться схожість поточної активності з патерном атаки (attack pattern similarity). Загальний вигляд цього розділу для підсистеми `prod-cluster-1` показано на рис. 3.8.

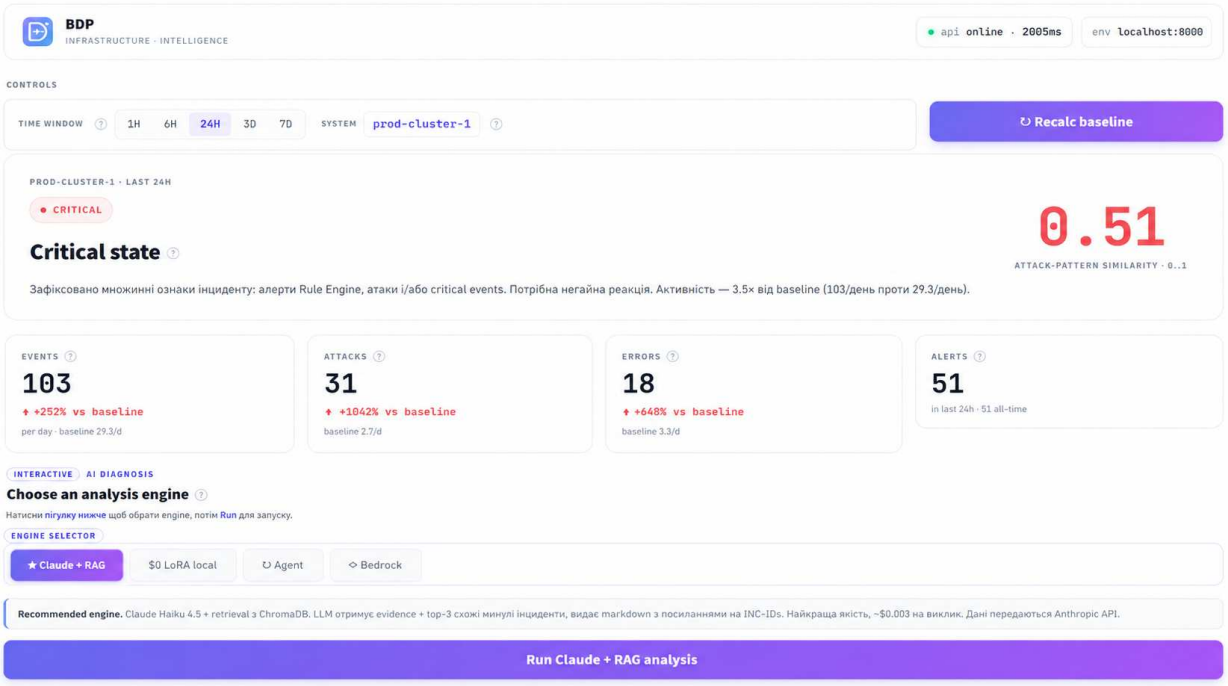


Рисунок 3.8 – Загальний вигляд дашборду оператора

Загальний вигляд інтерфейсу підтверджує реалізацію принципу «огляд → деталі»: у наведеному прикладі підсистема `prod-cluster-1` за останні 24 години має 103 події (+252 % від baseline), 31 атаку (+1842 %), 18 помилок (+448 %) та 51 сповіщення, тож система одразу позначає її станом `CRITICAL`. Усі ключові показники зібрані на одній сторінці, що дозволяє оператору швидко локалізувати проблемну підсистему та перейти до детального розгляду. Нижче розташовано розділ перегляду активності — інтерактивну часову шкалу розподілу подій за рівнем критичності (Critical, Warning, Info), побудовану на бібліотеці Plotly з підтримкою масштабування часового вікна та фільтрів за типом і рівнем події; над часовою шкалою виводиться зведення детермінованого рушія правил Rule Engine. Приклад відображення цього розділу наведено на рис. 3.9.

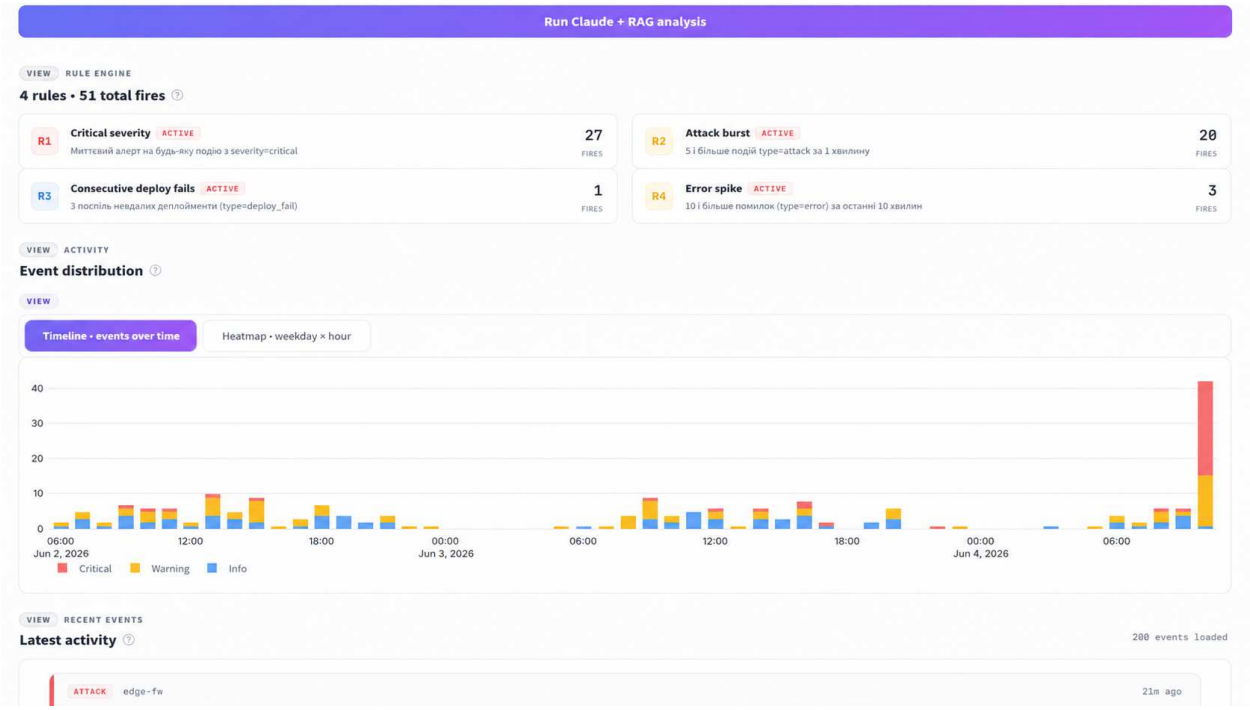


Рисунок 3.9 – Часова шкала розподілу подій за рівнем критичності та зведення рушія правил

Аномальну активність видно безпосередньо на часовій шкалі: різкий сплеск подій праворуч із переважанням сегментів critical (червоний) і warning (жовтий) візуально виділяється на тлі спокійного фону, а розташоване вище зведення Rule Engine показує, що спрацювали всі чотири правила (разом 51 спрацювання). Розділ AI Analysis Lab містить чотири окремі вкладки — по одній на кожен рушій AI-діагностики (Bedrock, Claude+RAG, LoRA local, function-calling agent). Кожна вкладка дозволяє запустити аналіз із власними параметрами та переглянути результат у форматі markdown. Картка Claude + RAG структурує діагностику у три секції — стан (Status), імовірна першопричина (Root Cause) і рекомендовані дії (Recommendations) — та наводить трійку найсхожіших історичних інцидентів, знайдених у векторному сховищі: `INC-0102`, `INC-0409` і `INC-0408` (тип `ddos_attack`, косинусна подібність у межах 0,75–0,77). Поряд відображаються метадані виклику (рушій, кількість токенів, вартість, латентність). Приклад картки результату наведено на рис. 3.10.

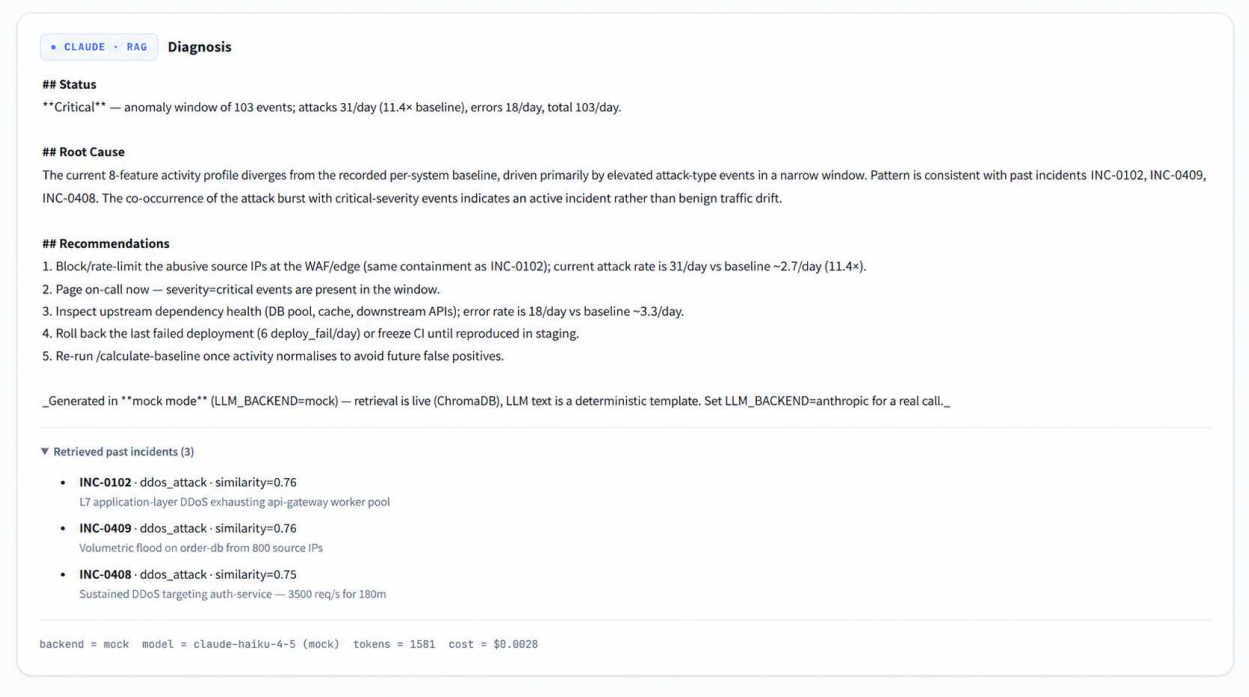


Рисунок 3.10 – Результат AI-діагностики у вкладці Claude + RAG

Для рушія function-calling агента додатково відображається повна траса виклику інструментів — у якому порядку агент звертався до `get_anomaly_score`, `get_baseline`, `get_recent_events`, `retrieve_similar_incidents` і завершального `final_answer`, та яких висновків доходив на кожному кроці (на наведеному прикладі — п’ять кроків). Це підвищує прозорість роботи агента для оператора, що особливо важливо при розборі складних або суперечливих діагностик. Приклад такої траси показано на рис. 3.11.

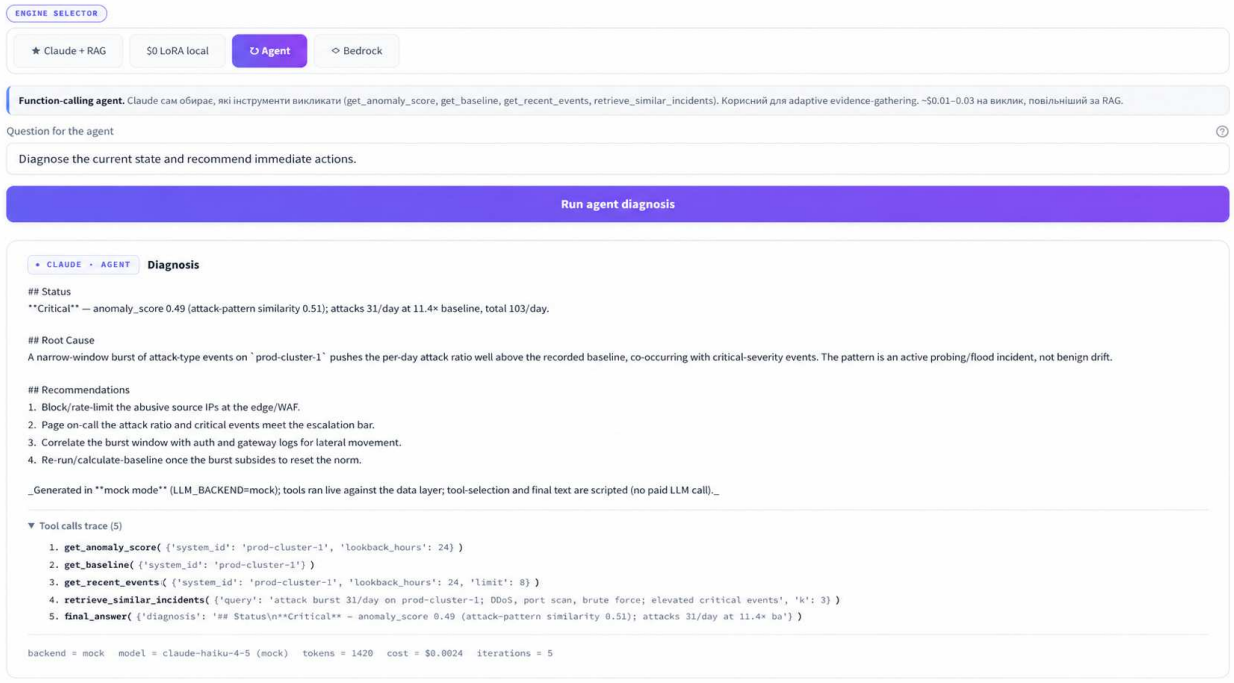


Рисунок 3.11 – Траса function-calling агента

Розділ статистики надає узагальнену кількісну інформацію по системі загалом: загальну кількість подій, розподіл за типами та рівнями критичності, вартість LLM-викликів за поточний місяць.

3.10 Розгортання у хмарних середовищах

Розгортання компонентів системи здійснюється у двох хмарних середовищах залежно від характеристик кожного компонента. Узагальнену схему розгортання наведено на рис. 3.12.

З рис. 3.12 видно, що автоматичне розгортання основного бекенду та дашборду з гілки main GitHub-репозиторію реалізовано через Railway, а ресурсоємкий LoRA-сервіс — на Azure Container Apps з режимом scale-to-zero. Такий поділ дозволяє платити повну вартість лише за активне використання дистильованої моделі (idle-вартість зведена до приблизно 5 USD на місяць за рахунок Azure Container Registry, що зберігає образ контейнера). Контроль

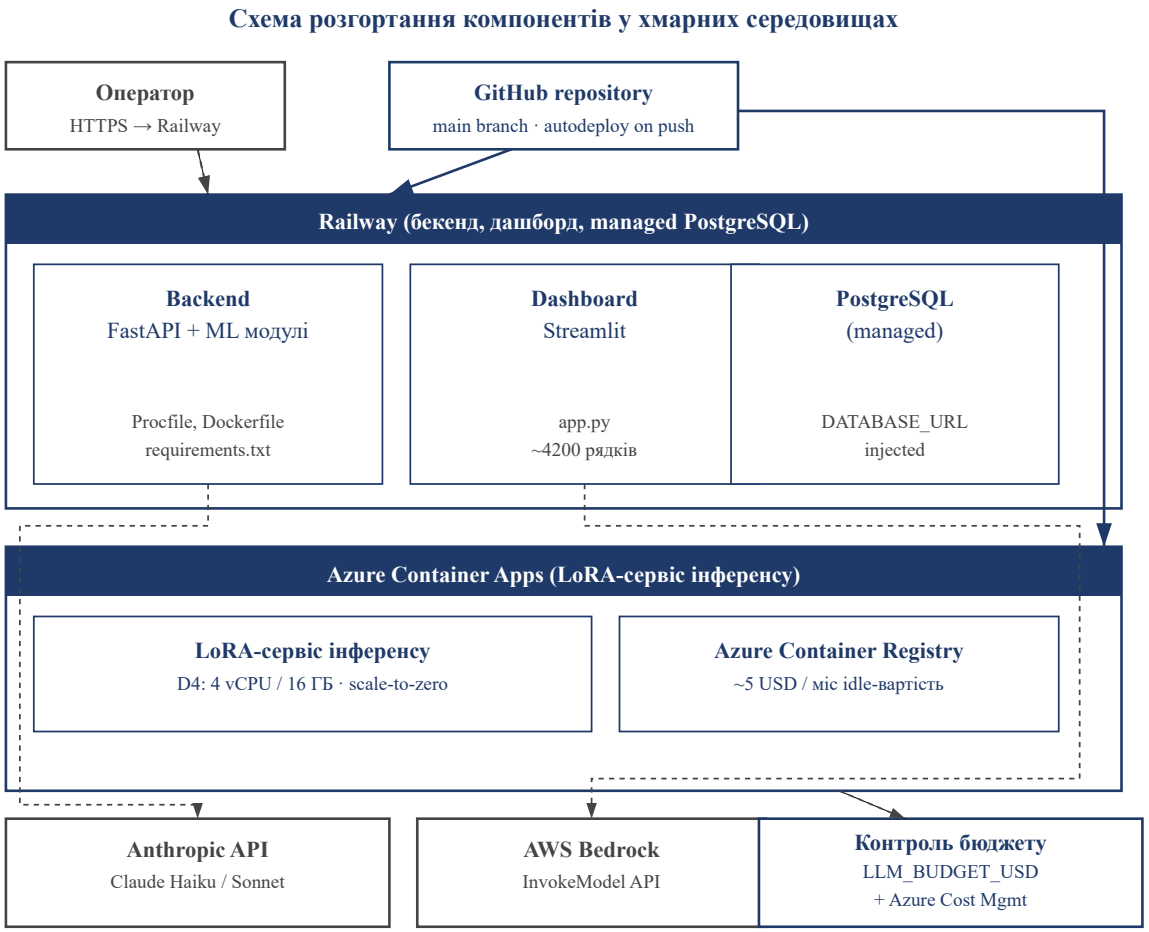


Рисунок 3.12 – Схема розгортання компонентів у хмарних середовищах

витрат на стороні AI-постачальників реалізовано у три рівні, що показано окремим блоком на схемі.

Основний бекенд та дашборд розгортаються на платформі Railway як два окремі сервіси з автоматичним розгортанням з гілки main GitHub-репозиторію. PostgreSQL надається як managed-послуга у тому ж проєкті Railway, що спрощує налаштування мережних з'єднань: Railway автоматично інжектує змінну оточення `DATABASE_URL` у контейнер бекенду. Конфігурація розгортання задається у файлах `Procfile` (для Railpack), `runtime.txt` (вказує версію Python) та `Dockerfile` (резервний шлях розгортання).

Сервіс LoRA-інференсу розгортається на Azure Container Apps з профілем навантаження D4 (4 vCPU, 16 ГБ оперативної пам'яті). Активовано режим scale-to-zero, при якому за відсутності запитів сервіс масштабується до нуля реплік, а при появі запиту автоматично піднімається протягом 30–90 секунд. Idle-вартість складає лише плату за Azure Container Registry (приблизно 5 USD на місяць). Розгортання LoRA-сервісу автоматизоване набором shell-скриптів у директорії `scripts/`: `deploy-azure-lora.sh` (одноразовий деплой), `azure-lora-on.sh` / `azure-lora-off.sh` / `azure-lora-status.sh` (керування життєвим циклом), `azure-set-budget.sh` (налаштування бюджетних сповіщень).

Контроль витрат реалізовано у три рівні: ліміт `LLM_BUDGET_USD` на рівні процесу бекенду (50 USD на місяць, atomic-резервація перед викликом моделі), місячний ліміт на стороні Anthropic API (через налаштування акаунту), бюджет з email-сповіщеннями на стороні Azure Cost Management (20 USD на місяць).

3.11 Вимоги до експлуатації та технічних засобів

Розроблений програмний комплекс розрахований на функціонування у хмарному середовищі під керуванням операційних систем сімейства Linux. Системні вимоги до апаратного забезпечення наведено у табл. 3.1.

Таблиця 3.1 – Мінімальні системні вимоги до компонентів системи

Компонент	**Мінімум CPU**	**Мінімум RAM**	**Дисковий простір**
Бекенд	2 vCPU	2 ГБ	5 ГБ (БД, моделі)
Дашборд оператора	1 vCPU	1 ГБ	1 ГБ
LoRA-сервіс	4 vCPU	16 ГБ	8 ГБ (образ з вагами)
PostgreSQL	1 vCPU	1 ГБ	залежить від обсягу даних

Вимоги до програмного забезпечення серверної сторони: ОС Linux (Ubuntu 22.04 LTS чи аналогічна), Python 3.11 або новіший, Docker 24.0 чи новіший (для контейнеризованого розгортання LoRA-сервісу), PostgreSQL 14 чи новіший [26]. Усі Python-залежності зафіксовано у файлах `requirements.txt` (основний бекенд), `requirements-dashboard.txt` (дашборд), `requirements-lora.txt` (LoRA-сервіс).

Вимоги до клієнтського робочого місця оператора: будь-який сучасний веб-браузер (Chrome 120+, Firefox 120+, Edge 120+, Safari 17+) з підтримкою HTML5 та JavaScript, роздільна здатність екрана не менше 1280×720, стабільне підключення до мережі Інтернет з пропускною здатністю не менше 5 Мбіт/с.

Вимоги до експлуатації включають регулярне оновлення Python-залежностей (рекомендовано раз на квартал) для отримання виправлень безпеки, моніторинг витрат через Anthropic Console та Azure Cost Management, регулярне резервне копіювання бази даних PostgreSQL (Railway виконує автоматичні щоденні резервні копії). Перенавчання моделі Isolation Forest рекомендовано виконувати при істотній зміні характеру навантаження (понад 20 % відхилення поточних метрик від часу тренування).

Безпека експлуатації забезпечується такими механізмами: обов'язкова автентифікація POST-ендпоінтів через заголовок `X-BDP-Key` з порівнянням через `hmac.compare_digest` для запобігання timing-атакам, шифрування трафіку на рівні транспорту через HTTPS (надається хмарним провайдером), регулярна ротація API-ключів, логування всіх операцій змінення стану. У цілому, експлуатаційні заходи відповідають рекомендаціям ДСТУ ISO/IEC 27001:2015 [3] та зведу практик ДСТУ ISO/IEC 27002:2015 [29].

3.12 Інструкція користувача

Інструкція користувача описує типові сценарії роботи оператора з розробленою системою. Для зручності сценарії згруповано за функціональними модулями дашборду.

3.12.1 Сценарій 1. Перегляд активності підсистем

Послідовність дій оператора така:

- відкрити браузер та перейти за URL дашборду;
- на головній сторінці у бічному меню обрати розділ «Огляд систем»;
- у таблиці підсистем переглянути показники: кількість подій за останні 24 години, кількість сповіщень, поточна оцінка аномальності;
- при необхідності клікнути на ім’я підсистеми для переходу до її детального перегляду.

Для оновлення даних використовується кнопка «Refresh» у правому верхньому куті дашборду. За замовчуванням дані кешуються на 60 секунд для зниження навантаження на бекенд.

3.12.2 Сценарій 2. Розслідування інциденту

При виявленні підвищеної оцінки аномальності або появи сповіщення оператор виконує таку послідовність дій:

- у розділі «Огляд систем» обрати підсистему з підвищеною аномальністю;
- у розділі «Timeline» переглянути часову шкалу подій з накладеним baseline-ом;
- звузити часовий діапазон до періоду навколо потенційного інциденту;

– за необхідності застосувати фільтр за типом події (attack, error, deploy_fail тощо);
– перейти у розділ «AI Analysis Lab» для отримання природномовної діагностики.

У розділі «AI Analysis Lab» оператор обирає один з чотирьох доступних рушіїв AI-діагностики:

– Bedrock baseline — швидка діагностика через AWS Bedrock без RAG;
– Claude + RAG — найвища якість діагностики з пошуком схожих історичних інцидентів;
– LoRA local — локальна діагностика без передавання даних за межі периметру;
– Function-calling agent — складна діагностика з адаптивним збором доказів.

Після вибору рушія оператор натискає кнопку «Run AI Diagnosis». Результат відображається у форматі markdown з трьома секціями: Status (Critical / Warning / Info), Root Cause (опис першопричини), Recommendations (конкретні дії для розв'язання). Поряд з результатом відображається метадата виклику: кількість токенів, вартість, латентність, перелік знайдених історичних інцидентів.

3.12.3 Сценарій 3. Перерахунок baseline-у підсистеми

При зміні характеру навантаження підсистеми оператор може запросити перерахунок базового профілю активності:

– у розділі деталей підсистеми натиснути кнопку «Recalculate baseline»;
– у формі обрати кількість днів історії для розрахунку (стандартно — 30 днів);

– підтвердити запит. Перерахунок виконується асинхронно, результат відображається після завершення;
– після завершення перерахунку оновлений baseline автоматично використовується у подальших обчисленнях оцінки аномальності.

3.12.4 Сценарій 4. Інтеграція з зовнішнім джерелом подій

Зовнішні системи (агенти моніторингу, сторонні застосунки) можуть надсилати події у систему через HTTP-ендпоінт `POST /events`. Формат запиту — JSON з обов'язковими полями timestamp (ISO 8601), source, type, severity, message, `system_id` та опційним полем metadata. У заголовку запиту необхідно вказати `X-BDP-Key` з виданим API-ключем. Приклад запиту:

```
POST /events HTTP/1.1
X-BDP-Key: <api-key>
Content-Type: application/json
{
  ”timestamp”: ”2026-04-25T10:30:00Z”,
  ”source”: ”web-server-01”,
  ”type”: ”error”,
  ”severity”: ”critical”,
  ”message”: ”Database connection timeout”,
  ”system_id”: ”prod-cluster-1”,
  ”metadata”: { ”database”: ”postgres-main” }
}
```

У разі успіху сервер повертає HTTP 201 Created з тілом, що містить ідентифікатор створеного запису. У разі помилки валідації — HTTP 422 з описом проблеми. У разі помилки автентифікації — HTTP 401.

ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі описано розробку серверного програмного комплексу. Реалізовано трирівневу архітектуру з чітким розділенням відповідальності між роутерами, сервісами та шарами доступу до даних.

Розроблено реляційну модель збереження даних на основі PostgreSQL та SQLAlchemy 2.0 з трьома основними таблицями (`events`, `alerts`, `baselines`) та підтримкою мультитенантності.

Реалізовано детермінований рушій правил R1–R4 для миттєвих сповіщень, ML-шар детекції аномалій на основі Isolation Forest з індивідуальним профілем активності за днями тижня та шар RAG на основі ChromaDB і sentence-transformers.

Реалізовано чотири взаємозамінні рушії AI-діагностики (Bedrock baseline, Claude + RAG, LoRA local student, function-calling agent), що повертають однотипну структуровану відповідь.

Реалізовано шар захисту від prompt-injection за принципом глибокого захисту з п’ятьма механізмами: регулярні вирази, UUID-обгортання, детектор кодувань, інструкції системного промпта, обмеження прав агента.

Виконано дистиляцію знань teacher-моделі (Claude Haiku) у локальну student-модель (Qwen2.5-1.5B + LoRA) технікою QLoRA на GPU T4.

Розроблено інтерактивний дашборд оператора на основі Streamlit. Розгорнуто компоненти у хмарних середовищах Railway та Azure Container Apps з трирівневим контролем витрат.

Сформульовано вимоги до експлуатації та технічних засобів, розроблено інструкцію користувача з чотирма сценаріями: перегляд активності підсистем, розслідування інциденту, перерахунок baseline, інтеграція з зовнішнім джерелом подій.

РОЗДІЛ 4
ВИПРОБУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1 Методологія оцінювання

Для забезпечення коректності та відтворюваності експериментальних досліджень розроблено уніфіковану методологію оцінювання, що застосовується до всіх компонентів системи. Методологія базується на трьох принципах: повна відтворюваність (фіксація усіх random seed та конфігураційних параметрів), використання відкритих наукових корпусів (HDFS_v1 [7; 19], CICIDS2017 [8]) для забезпечення коректності порівняння з опублікованими результатами, чесна фіксація негативних результатів (експерименти, де очікуваний ефект не підтвердився, не приховуються, а документуються разом з обговоренням можливих причин).

Для задач supervised класифікації використано стандартний pipeline: стратифікований розподіл вибірки 80/20 з фіксованим random_state = 42, тренування трьох моделей (Logistic Regression, Random Forest, XGBoost+SMOTE [17; 18]), тюнінг порогу класифікації методом 5-fold cross-validation з оптимізацією за F1-мірою, оцінка за метриками accuracy, precision, recall, F1, ROC-AUC. Для багатокласових задач застосовано per-class threshold tuning і звітування macro-F1 та weighted-F1 окремо.

Для задачі оцінювання якості AI-діагностики застосовано підхід LLM-as-judge: відповіді кожного з рушіїв оцінює окрема модель-суддя (Claude Haiku 4.5) за чотирма показниками — `factual_accuracy` (фактична достовірність), actionability (придатність рекомендацій для виконання), completeness (повнота охоплення доказів), `hallucination_risk` (ризик галюцинацій моделі). Підсумкова метрика overall обчислюється як зважене

середнє цих чотирьох показників. Цей підхід дозволяє автоматично оцінювати десятки сценаріїв без потреби в ручному expert-аналізі кожної відповіді.

Для тестування стійкості до prompt-injection атак сформовано корпус з 27 атак, що містить 17 класичних шаблонів (взятих з відкритих публікацій з безпеки LLM [27]) та 10 доменно-специфічних розширень (адаптованих до контексту моніторингу безпеки інфраструктури). Кожна атака виконується у двох умовах — без захисних механізмів та з активованими guardrails.

4.2 Дослідження класифікаторів

Дослідження класифікаторів виконано на трьох незалежних задачах: класифікація рівня критичності події (severity), класифікація аномалій на корпусі HDFS_v1, класифікація типу атаки на корпусі CICIDS2017. Метою цих експериментів було оцінити придатність класичних supervised методів для трьох різних типів задач виявлення інцидентів, а також дослідити, чи дають моделі чесний семантичний сигнал, або просто відтворюють шаблонні правила розмітки.

Експеримент 1 — severity baseline. Тренувальний набір сформовано з 67 784 подій, отриманих об’єднанням синтетичного корпусу з корпусами HDFS_v1 та CICIDS2017. Усі три моделі досягли практично ідентичних показників якості: XGBoost+SMOTE — accuracy 0,9976, macro-F1 = 0,9978, ROC-AUC = 1,000. Аналіз важливості ознак показав, що модель фактично відтворює детерміноване правило розмітки парсера, оскільки severity у вихідних даних походить від type через детерміноване перетворення. Цей експеримент включено у роботу як методологічний baseline, що демонструє ризики supervised постановки на похідних мітках: висока точність може бути обумовлена не виявленням реального семантичного сигналу, а простим відтворенням правила розмітки.

Експеримент 2 — детекція аномалій на HDFS_v1. Корпус HDFS_v1 з 10 000 подій містить людські мітки аномалій (роботи Xu et al. [7]), отримані шляхом ручного розмічування експертами. Це справді semantic ground truth, що не виводиться з інших полів запису. На цьому корпусі найкращий результат показала модель XGBoost+SMOTE: accuracy 0,7550, F1 = 0,7118, ROC-AUC = 0,8086. Цей результат відповідає рівню, опублікованому в науковій літературі для supervised підходів на цьому корпусі. Аналіз найважливіших ознак виявив семантичні маркери (tfidf_datanode dataxceiver, tfidf_blockinfo, kw_error), що корелюють з аномальним життєвим циклом блоку HDFS, а не з самою міткою — отже, модель виявляє реальний семантичний сигнал, а не артефакт розмітки. Це істотно: F1 = 0,71 на корпусі з людською розміткою — чесний, академічно цінний результат.

Експеримент 3 — класифікація типу атаки на CICIDS2017 [8] (з виправленими мітками за роботою Engelen et al. [32], що містить аналіз label leakage у оригінальному корпусі). Корпус з 22 137 подій після згортання 19+ оригінальних класів атак у 7 сімейств (BENIGN, DDoS_DoS, PortScan, BruteForce, WebAttack, Botnet, Infiltration). Найкращий результат — XGBoost+SMOTE: accuracy 0,9995, macro-F1 = 0,9996, ROC-AUC = 1,000. Аналіз важливості ознак показав домінування мережевих артефактів (порти 80, 443, 8080, протоколи UDP/TCP), що підтверджує: метадані потоків достатні для класифікації типу атаки без аналізу пакетів. Висока точність на цьому корпусі узгоджується з опублікованими результатами і пояснюється тим, що кожен тип атаки має характерну port/protocol-сигнатуру (DDoS типово використовує HTTP/HTTPS, PortScan послідовно сканує діапазон портів, BruteForce націлений на порти SSH чи FTP) [16]. Окремо наголошуємо, що навіть після виправлень за [32] корпус CICIDS2017 зберігає певний ступінь port/protocol-кореляцій, який спрощує задачу для дерев рішень.

Узагальнені кількісні результати трьох експериментів наведено у табл. 4.1, а їх графічну візуалізацію для двох корпусів з людськими мітками — на рис. 4.1.

Таблиця 4.1 – Результати трьох експериментів класифікації

Експеримент	**Найкраща модель**	**Accuracy / F1**	**ROC-AUC**	**Тип міток**
Severity baseline	XGBoost+SMOTE	0,998 / 0,998	1,000	Похідні (правило)
HDFS anomaly	XGBoost+SMOTE	0,755 / 0,712	0,809	Людські (SOSP 2009)
CICIDS attack family	XGBoost+SMOTE	0,999 / 0,9996	1,000	Людські (ICISSP 2018)

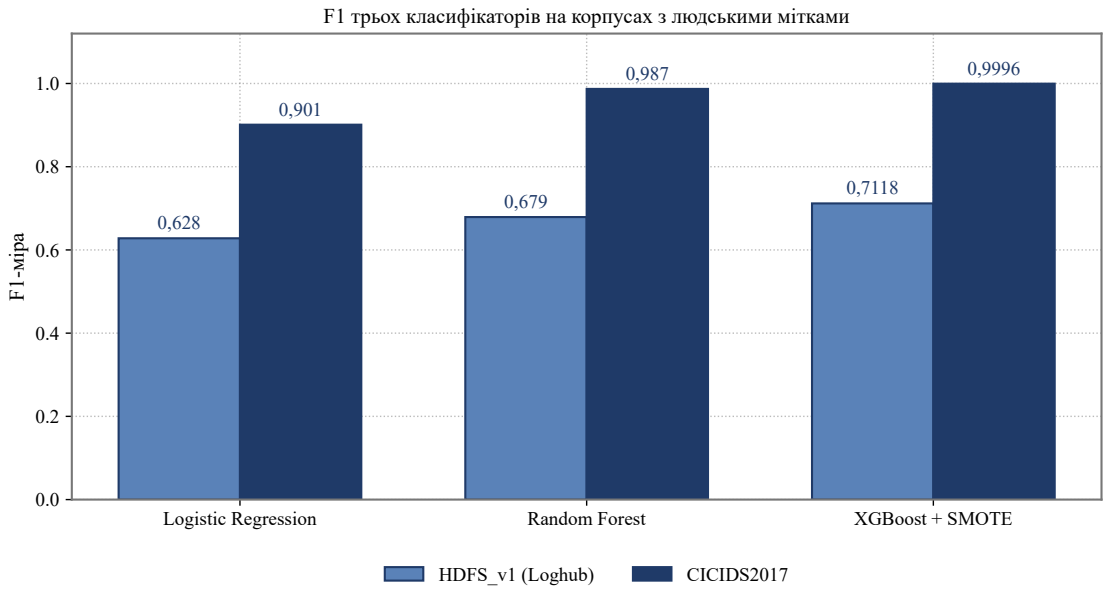


Рисунок 4.1 – F1-міри трьох класифікаторів на корпусах HDFS_v1 і CICIDS2017

З рис. 4.1 видно, що XGBoost+SMOTE стабільно перевершує Logistic Regression та Random Forest на обох корпусах: на HDFS_v1 розрив до RF становить близько 0,03 в F1-мірі, до LR — близько 0,08; на CICIDS2017 усі три методи демонструють F1 $\geq$ 0,90, але XGBoost наближається до 1,0. Які-

сна різниця між корпусами пояснюється природою міток: на HDFS_v1 людська ручна розмітка містить нетривіальні гранично-важкі приклади, тоді як на CICIDS2017 кожен тип атаки залишає характерну port/protocol-сигнатуру, що тривіально вловлюється деревами рішень. Цей результат підтверджує доцільність вибору XGBoost+SMOTE як основного класифікатора для системи.

На основі трьох експериментів можна зробити такі висновки. По-перше, для задач з людськими мітками (HDFS, CICIDS) метод XGBoost+SMOTE стабільно показує найкращу якість, що підтверджує доцільність його вибору як основного класифікатора. По-друге, F1 = 0,71 на HDFS — це чесний science-result, а не свідчення слабкості системи: на цьому корпусі практично всі опубліковані supervised-методи показують схожий рівень [19]. По-третє, методологічна обережність при формуванні корпусу (виявлення похідних міток) критично важлива для інтерпретації високих метрик.

4.3 Дослідження ефективності шару RAG

Для оцінки впливу шару RAG на якість AI-діагностики проведено A/B-експеримент на 10 фіксованих сценаріях з різними патернами інцидентів (DDoS, brute-force, deploy failure, intrusion suspect, data corruption, port scan, mixed signals тощо). Кожен сценарій оброблявся в двох умовах: без RAG (baseline) та з RAG. Якість оцінювалася моделлю-суддею Claude Haiku 4.5 за чотирма показниками [20].

Початкова конфігурація RAG (інжектування топ-3 інцидентів без фільтрації) показала несподіваний негативний результат: загальний Δ overall склав $-0{,}055$, тобто RAG погіршив якість діагностики. Аналіз показав, що ризик галюцинацій зріс з 0,102 до 0,200, що пояснюється некритичним запозиченням специфічних деталей з історичних інцидентів (наприклад, кон-

кретних номерів версій програмного забезпечення, IP-адрес, ідентифікаторів реплік), що не відповідають поточному контексту.

Найгірший випадок — сценарій deploy_failure з падінням overall з 0,868 до 0,582 (−0,287). Аналіз цього кейсу показав, що retrieval коректно знайшов схожі історичні інциденти INC-0259 та INC-0168 з подібністю 0,68–0,75, проте модель цитувала їх специфічні деталі (зокрема, конкретні версії v2.6.12 та v3.9.14), яких не було ні у поточних доказах, ні в знайденому контексті — отже, модель галюцинувала версії, які звучали правдоподібно у контексті посилань на історичні інциденти.

Для виправлення ситуації внесено дві зміни. По-перше, додано поріг подібності `min_score` = 0,70, нижче якого знайдені інциденти відкидаються. По-друге, переформульовано системний промпт: знайдені інциденти подаються як «loose pattern hints, NOT confirmed analogues» з явною забороною запозичувати специфіку.

Кожна з двох конфігурацій RAG (v1 і v2) була запущена окремим прогоном з власним baseline-прогоном без RAG на тих самих 10 сценаріях; внутрішня варіація моделі-судді та малий розмір корпусу зумовили те, що абсолютні значення baseline двох прогонів різняться (overall 0,828 у прогоні v1 проти 0,856 у прогоні v2). Тому для коректного висновку про вплив шару RAG використовується $\Delta = \text{RAG} - \text{baseline}$ у межах кожного прогону окремо, а не порівняння абсолютних значень між прогонами. У табл. 4.2 відображено обидві пари baseline / RAG із відповідними Δ, а графічне порівняння — на рис. 4.2.

Таблиця 4.2 – Порівняння конфігурацій шару RAG (кожна v порівнюється з власним baseline)

Показник	**Baseline v1**	**RAG v1**	**Δ v1**	**Baseline v2**	**RAG v2**	**Δ v2**
`factual_accuracy`	0,874	0,825	−0,049	0,905	0,881	−0,024
actionability	0,861	0,821	−0,040	0,861	0,860	−0,001
completeness	0,841	0,854	+0,013	0,864	0,869	+0,005
`hallucination_risk`	0,102	0,200	+0,098	0,070	0,100	+0,030
overall	0,828	0,773	−0,055	0,856	0,840	−0,016

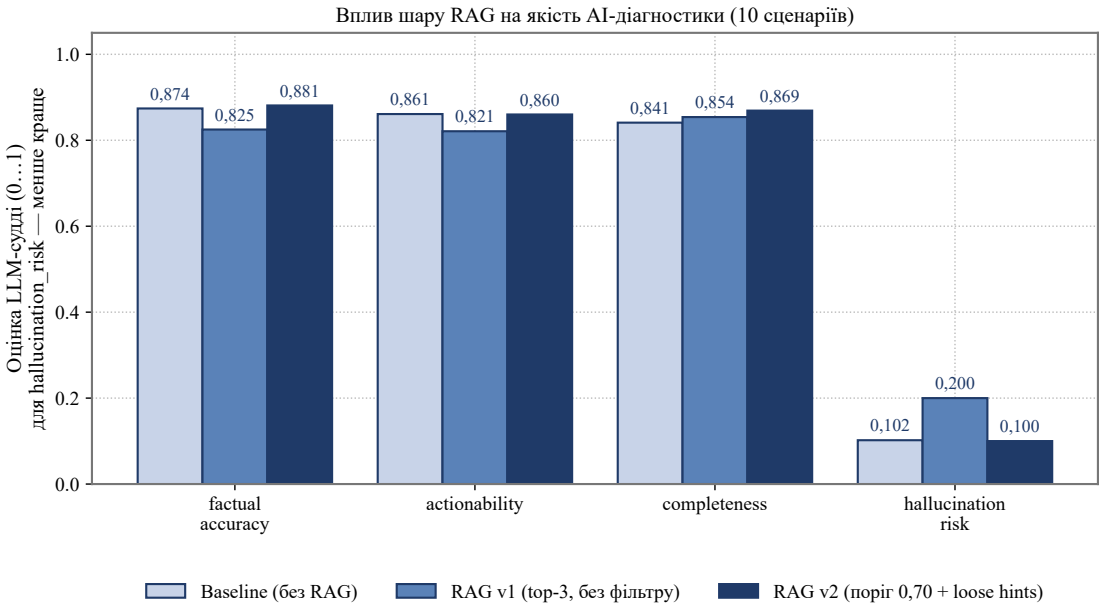


Рисунок 4.2 – Вплив шару RAG на чотири показники якості діагностики

На рис. 4.2 показано абсолютні значення метрик для трьох конфігурацій (Baseline, RAG v1, RAG v2). Через відмінність baseline між прогонами читати рисунок треба у поєднанні з табл. 4.2, де колонка Δ обчислена коректно — RAG − baseline у межах одного й того самого прогону. Після оптимізацій ризик галюцинацій знижено майже втричі (Δ `hallucination_risk`: з +0,098 у v1 до +0,030 у v2), а загальний розрив (Δ overall) — у три з половиною рази (з −0,055 у v1 до −0,016 у v2). Конфігурація v2 (поріг 0,70 +

переформульовані інструкції) не досягає чистого позитивного ефекту, але наближається до точки беззбитковості, що дозволяє зберегти оператору доступ до релевантних історичних інцидентів без істотної шкоди для якості діагностики.

У трьох з десяти сценаріїв жоден з знайдених інцидентів не подолав поріг 0,70 — у цих випадках RAG став ідентичним до baseline ($\Delta \approx 0$). Хоча у фінальній конфігурації RAG не досягає чистого позитивного ефекту на цьому корпусі, документується це як чесний негативний висновок з обґрунтуванням можливих напрямків подальшого вдосконалення: збільшення різноманіття корпусу інцидентів через генерацію LLM (а не шаблонами), переранжування знайдених кандидатів крос-енкодером, ablation на розмірі корпусу. Цей результат є важливим методологічним внеском роботи: показано, що наївна реалізація RAG може погіршити якість діагностики, і запропоновано конкретні механізми мітигації, що повертають якість майже у точку беззбитковості.

4.4 Дослідження стійкості до prompt-injection атак

Стійкість шару LLM-діагностики до prompt-injection атак досліджено на корпусі з 27 атак, що містить 17 класичних шаблонів (взятих з відкритих публікацій з безпеки LLM [27]) та 10 доменно-специфічних розширень. Кожна атака виконувалася у двох умовах: без захисних механізмів (unguarded) та з активованими guardrails (guarded). Результати оцінювалися моделлю-суддею за критерієм «hijacked = true/false» — чи вдалося атаці перехопити поведінку моделі. Узагальнені результати наведено у табл. 4.3.

Таблиця 4.3 – Результати red-team тестування шару LLM-діагностики

Метрика	**Без захисту**	**Із захистом**
Judge hijack rate (основний показник стійкості, оцінка LLM-суддею)	0 / 27 (0,0 %)	0 / 27 (0,0 %)
Defensive markers у відповіді моделі	25 / 27 (92,6 %)	27 / 27 (100,0 %)
Допоміжний keyword-індикатор (лексична евристика)	10 / 27 (37,0 %)	9 / 27 (33,3 %)
Вартість прогону моделі-судді	$0,0199	$0,0205

Як видно з табл. 4.3, шар захисту досягає двох основних цілей. Основним показником стійкості є judge hijack rate — частка атак, після яких LLM-суддя визнав, що модель виконала зловмисну інструкцію замість аналітичного завдання. Значення judge hijack rate складає 0 / 27 (0,0 %) як без захисту, так і з активованими guardrails: жодна атака не перехопила поведінку моделі. Це підтверджує семантичну стійкість шару діагностики до досліджених прийомів prompt-injection. Друга метрика — defensive markers (наявність у відповіді моделі явного відмовлення виконати ін’єкційну команду) — зростає з 92,6 % до 100,0 % при увімкненні захисту: усі без винятку відповіді тепер містять видимий слід відмови, що критично важливо для аудиту, telemetry і регуляторної перевірки.

Третій рядок таблиці — keyword-індикатор — є допоміжною лексичною евристикою і наводиться лише для повноти. Він рахує входження певних поверхневих маркерів у вихідному тексті відповіді моделі (наприклад, слів зі словника атак, що повторилися у відмові моделі або у її поясненні факту виявлення спроби атаки). Цей індикатор схильний до значного false-positive-шуму: характерні слова можуть з’являтися у цілком безпечних відповідях моделі саме тому, що вона коректно ідентифікує і обговорює спробу атаки. Тому показник keyword у 10 / 27 без захисту та 9 / 27 з захистом не вимірює стійкість системи, а лише відображає природне коливання лексичних збі-

гів у двох прогонах. Робити з різниці в одну атаку (10 проти 9) висновок про неефективність захисту методологічно некоректно — статус стійкості визначається судом LLM-судді й рівнем defensive markers, а не цією допоміжною евристикою.

Таким чином, шар захисту виконує свою функцію: судом моделі-судді жодна з 27 атак не перехопила поведінку (0 / 27 в обох умовах), а сама модель тепер у 100 % випадків явно сигналізує про виявлену спробу ін'єкції (зростання з 92,6 % до 100,0 % defensive markers). Розширений захист (UUID-обгортання контенту та явні правила у системному промпті, що належать до Рівнів 2 і 4 на рис. 3.6) забезпечує додатковий шар оборони на випадок атак, які могли б обходити вбудовану стійкість моделі у майбутніх ітераціях.

4.5 Дослідження ефективності function-calling агента

Для оцінки доцільності використання function-calling агента порівняно з single-shot діагностикою проведено A/B-експеримент на 10 фіксованих сценаріях. Кожен сценарій обробляв single-shot RAG-pipeline (baseline) та function-calling агент із заглушеними інструментами, що повертали ті самі дані, які single-shot pipeline отримував детерміновано. Це ізолює змінну, яку ми хочемо виміряти (вплив агентного циклу) від інших чинників (різниця у даних).

Таблиця 4.4 – Порівняння single-shot pipeline та function-calling агента

Показник	**Single-shot RAG**	**Agent**	**Δ (agent – baseline)**
`factual_accuracy`	0,881	0,891	+0,010
actionability	0,861	0,768	−0,093
completeness	0,866	0,831	−0,035
`hallucination_risk`	0,127	0,117	−0,010
overall	0,831	0,795	−0,036
Вартість 10 кейсів	$0,029	$0,096	× 3,3

Function-calling агент не перевершив single-shot pipeline на сценаріях з фіксованими доказами: загальний Δ overall = −0,036 на користь baseline. Найбільший розрив — на показнику actionability (−0,093), що пояснюється тим, що агент іноді витрачає ітерації циклу на додатковий збір контексту, в результаті виходить менше токенів на формування конкретних рекомендацій. При цьому вартість виклику агента в середньому в 3,3 рази вища за рахунок ітеративного циклу взаємодії з моделлю (середнє число ітерацій — 2,5 на сценарій, з кожною ітерацією супроводжується 4–5 викликами інструментів).

Цей результат включено у роботу як чесний негативний висновок: для діагностики з заздалегідь зібраними доказами single-shot pipeline є оптимальним вибором. Function-calling агент проявляє переваги в інших сценаріях — змінній глибині запитів («порівняти останні 24 години та останній тиждень»), мульти-системних запитах («яка з моїх систем зараз найгарячіша?»), вільних запитах оператора, де `system_id` та lookback не задані заздалегідь. Ці сценарії не входять у поточний бенчмарк з 10 кейсів, тому результат є чесною оцінкою агента у вузькій ніші — не його справжніх потенційних можливостях. Чесний звіт про межі застосовності методу — важливий методологічний внесок роботи.

4.6 Дослідження ефективності дистильованої моделі

Для оцінки якості дистиляції знань з teacher-моделі (Claude Haiku 4.5) у student-модель (Qwen2.5-1.5B + LoRA, r=16 [24; 25]) проведено два незалежні A/B-прогони: на curated-наборі з 10 ретельно сформованих сценаріїв (відомих заздалегідь, ручне формування) та на randomised held-out наборі з 10 випадково згенерованих сценаріїв (фіксований seed = 24601, той самий розподіл архетипів, що й тренувальний набір, але інші конкретні приклади). Цей подвійний A/B-прогон є найсильнішим тестом на overfitting: якщо результати на двох наборах істотно розходяться, це вказує на прихований підгін під curated-кейси.

У табл. 4.5 наведено результати обох прогонів, а на рис. 4.3 — їх графічну візуалізацію.

Таблиця 4.5 – Результати двох A/B-прогонів дистиляції

Показник	**Teacher curated**	**Student curated**	**Teacher held-out**	**Student held-out**
`factual_accuracy`	0,888	0,784	0,868	0,755
actionability	0,877	0,764	0,854	0,781
completeness	0,869	0,802	0,834	0,784
`hallucination_risk`	0,112	0,220	0,115	0,270
overall	0,844	0,717	0,818	0,692

На рис. 4.3 відмінно видно, що student-модель послідовно відтворює близько 85 % якості teacher-моделі на обох прогонах: 0,717 / 0,844 = 0,849 на curated-наборі та 0,692 / 0,818 = 0,846 на held-out-наборі. Розбіжність між двома прогонами становить лише 0,002 в overall-метриці, що є найсильнішим підтвердженням того, що результат дистиляції узагальнюється на нові сценарії і не є наслідком прихованого підгону під curated-кейси. Це дозволяє

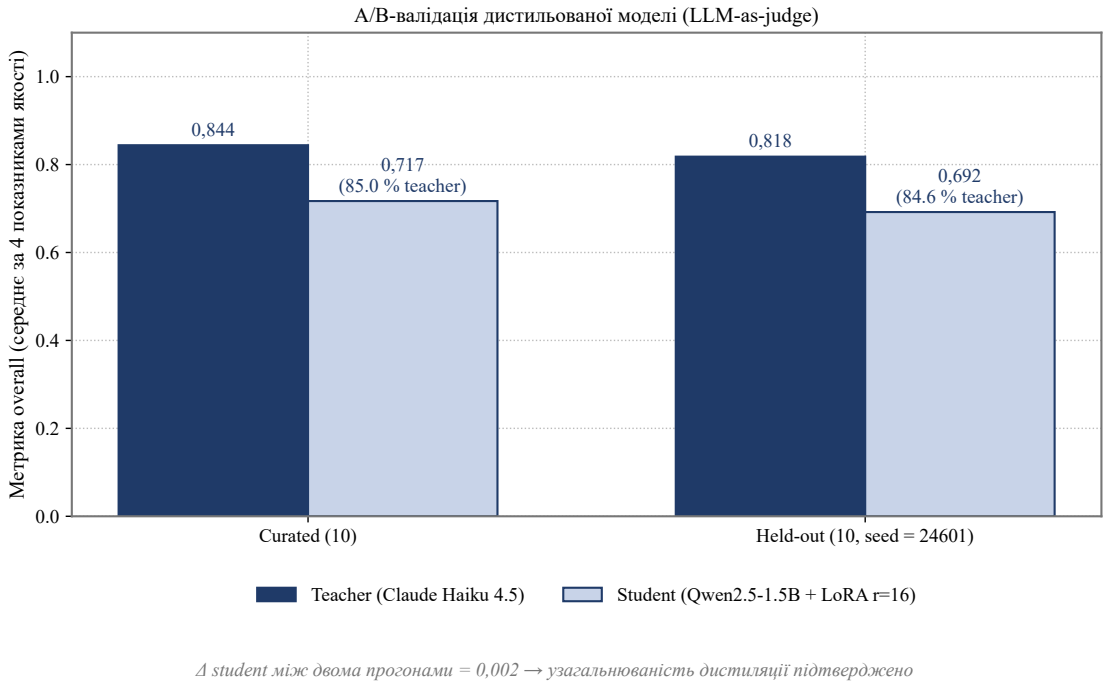


Рисунок 4.3 – A/B-валідація дистильованої моделі на двох незалежних прогонах

з високою впевненістю стверджувати, що метод дистиляції (QLoRA r=16, 200 пар, 3 епохи) дійсно передає знання teacher-моделі у student-модель, а не просто запам'ятовує конкретні приклади з тренувального набору.

Підвищений рівень `hallucination_risk` у student-моделі (0,220 на curated, 0,270 на held-out) проти стабільного teacher-рівня (0,112 / 0,115) вказує на основний слабкий бік дистильованої моделі — періодичне вигадування специфічних деталей. Зокрема, приріст 0,050 між двома student-прогонами свідчить, що при роботі з раніше не баченими сценаріями ризик галюцинацій додатково зростає. Це типовий ефект малого тренувального набору (200 пар) і може бути зменшений збільшенням корпусу до 500–1000 пар у майбутніх ітераціях. Аналогічно, латентність на M-series CPU виявилася значно вищою за прогноз (близько 58 секунд на виклик) через обмеження Apple MPS backend на максимальний розмір ndarray; на T4 GPU генерація займає близько 1 секунди.

Кількісно дистильована модель забезпечує 85 % якості teacher-моделі при нульовій маргінальній вартості виклику (без врахування фіксованої idle-вартості Azure Container Registry). Це робить її оптимальним вибором для трьох сценаріїв: privacy-sensitive обробки даних (повна локальність), cost-sensitive deployments (нульова маргінальна вартість), air-gapped environments (відсутність потреби у зовнішніх API-викликах).

Підсумково, здатність системи виявляти реальні аномалії підтверджується на робочому навантаженні підсистеми `prod-cluster-1`: на ньому спрацювали всі чотири правила детермінованого рушія Rule Engine (R1 = 27, R2 = 20, R3 = 1, R4 = 3; разом 51 спрацювання). Це дає детермінований доказ аномалії, який не залежить від порогів ML-моделі та підтверджує її незалежним сигналом. Зведення спрацювань правил для цього випадку показано на рис. 4.4 (повний вигляд часової шкали тих самих подій наведено в розд. 3 на рис. 3.9).

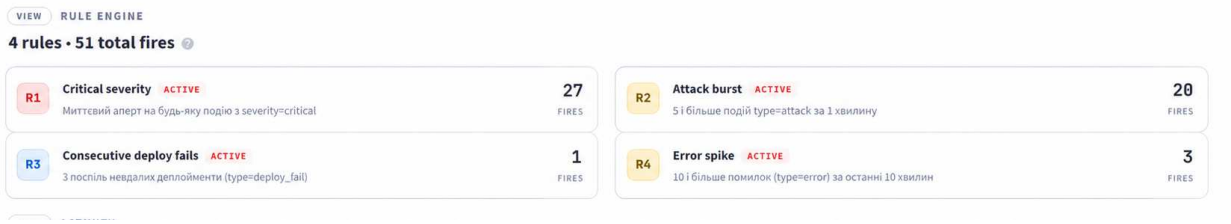


Рисунок 4.4 – Спрацювання правил Rule Engine при детекції реальної аномалії

4.7 Аналіз експлуатаційної вартості системи

Окрім якісних показників, важливою характеристикою розробленого програмного продукту є його операційна вартість — як на стадії експериментального налагодження, так і у штатному режимі експлуатації. У табл. 4.6 наведено розподіл сукупних витрат на виклики LLM API під час експериментальної фази за етапами.

Таблиця 4.6 – Розподіл витрат на виклики LLM API за фазами

Етап	**Опис**	**Вартість, USD**
Phase 2	RAG smoke + початкове тестування	≈ 0,16
Phase 3	RAG A/B-прогони (v1 + v2)	≈ 0,17
Phase 4	Red-team прогони (2 ітерації)	≈ 0,17
Phase 5	Function-calling agent A/B	≈ 0,15
Phase 6	Дистиляція (генерація 200 пар)	≈ 0,60
Phase 6	LoRA A/B-прогони (curated + held-out)	≈ 0,12
Загалом	Сукупні витрати на виклики LLM API	≈ 1,37

З табл. 4.6 видно, що домінуючий внесок у сумарну вартість зробила фаза генерації корпусу для дистиляції (приблизно 0,60 USD = майже половина від загального бюджету). Це очікувано, оскільки на цій фазі для кожного з 200 елементів корпусу виконувався виклик teacher-моделі (Claude Haiku 4.5) з повним промптом-доказами. Решта фаз розподілені рівномірно у межах 0,15–0,17 USD, що відповідає 10–12 викликам моделі на кожну з них. Сукупні витрати на виклики LLM API під час експериментальної фази склали приблизно 1,37 USD, що є важливим практичним результатом, що демонструє економічну доступність експериментального дослідження сучасних AI-систем у академічному середовищі.

Стосовно штатної експлуатації після завершення налагодження, очікувані характеристики вартості такі: для рушія LoRA local — нульова маргінальна вартість виклику при ввімкненому сервісі (idle вартість ≈ 5 USD/місяць за рахунок Azure Container Registry); для рушіїв Claude+RAG та Bedrock — приблизно 0,003 USD за виклик; для function-calling агента — приблизно 0,01–0,03 USD за виклик.

Для порівняння: в межах експериментального бюджету (1,37 USD) можна провести десятки A/B-прогонів якості, ablation-експерименти, red-team тестування. Це принципово знижує бар’єр входу для академічних дослідників та дозволяє студенту виконати повноцінне дослідження з декількох фаз у межах академічного проєкту.

ВИСНОВКИ ДО РОЗДІЛУ 4

У четвертому розділі викладено результати п’яти експериментальних досліджень розроблених компонентів системи.

Дослідження класифікаторів на трьох незалежних задачах підтвердило адекватність методології supervised класифікації: на корпусі HDFS_v1 з людськими мітками досягнуто F1 = 0,712, що відповідає опублікованим результатам [7; 19], на корпусі CICIDS2017 — F1 = 0,9996 [8; 32]. Експеримент з severity baseline продемонстрував важливість методологічної обережності: висока точність може приховувати просте відтворення правила розмітки.

Дослідження шару RAG показало, що наївне інжектування знайдених інцидентів погіршує якість діагностики через зростання галюцинацій (Δ overall = $-0{,}055$ у початковій конфігурації v1). Дві оптимізації — поріг подібності 0,70 та переформулювання системного промпта — знизили розрив до Δ overall = $-0{,}016$ у конфігурації v2 (розрахунок Δ — у межах одного прогону, кожен зі своїм baseline). Це наближає шар RAG до точки беззбитковості, проте чистого позитивного ефекту на даному корпусі не досягнуто. Документується це як чесний негативний висновок з обґрунтуванням напрямків подальшого вдосконалення.

Дослідження стійкості до prompt-injection атак показало, що сучасна модель Claude Haiku 4.5 не була перехоплена жодною з 27 атак як без захисту, так і з захистом [27]. Захисний шар забезпечив повне покриття виявлення атак (зростання defensive markers з 92,6 % до 100 %), що є важливим для аудиту.

Дослідження function-calling агента показало, що для діагностики з заздалегідь зібраними доказами single-shot pipeline є оптимальним вибором: агент програв на 0,036 в overall-метриці при втричі вищій вартості. Цей ре-

зультат включено як чесний негативний висновок з визначенням сфер, де переваги агента можуть бути продемонстровані у подальших дослідженнях.

Дослідження дистильованої моделі підтвердило, що student-модель Qwen2.5-1.5B з LoRA-адаптером відтворює 85 % якості teacher-моделі на двох незалежних A/B-прогонах [24; 25]. Збіжність двох прогонів у межах 0,002 є найсильнішим підтвердженням узагальнюваності результатів дистиляції.

Сукупні витрати на виклики LLM API під час експериментальної фази склали приблизно 1,37 USD. У штатній експлуатації після завершення налагодження маргінальна вартість виклику дистильованої моделі є нульовою (за наявності ввімкненого LoRA-сервісу), а для рушіїв на основі зовнішніх API — у межах 0,003–0,03 USD за виклик. Отримані оцінки демонструють економічну доступність експериментального дослідження та штатної експлуатації сучасних AI-систем у академічному середовищі.

ВИСНОВКИ

У дипломному проєкті розроблено серверний програмний комплекс «Система збору та аналізу сигналів безпеки комп'ютерної інфраструктури», що вирішує задачу автоматизованого виявлення аномалій безпеки з природномовною діагностикою інцидентів. Розроблений програмний продукт належить одночасно до трьох категорій згідно з §5 Положення про виконання дипломних проєктів кафедри обчислювальної техніки [5]: системи аналізу й обробки даних (основна категорія), системи знань (через шар RAG та дистильовану локальну модель) та системи прогнозування (через статистичну модель Isolation Forest).

Порівняння розробленого програмного продукту з існуючими аналогами (Splunk Enterprise Security, Elastic Security, Wazuh, Datadog Security Monitoring) дозволяє виокремити такі переваги: відсутність ліцензійних відрахувань (повністю відкритий технологічний стек), наявність вбудованої природномовної діагностики з підкріпленням контекстом схожих історичних інцидентів (відсутня у всіх чотирьох розглянутих рішеннях [10]), можливість локального інференсу для забезпечення приватності оброблюваних даних [1; 3], відносно низькі мінімальні вимоги до апаратного забезпечення (2 vCPU / 2 ГБ RAM для основного бекенду). Обмеженням системи порівняно з промисловими SIEM є менша база попередньо налаштованих кореляційних правил (4 базові правила R1–R4 проти тисяч у Splunk та Wazuh), що компенсується інтерпретованим виходом ML-шару та шару LLM-діагностики.

У ході виконання роботи отримано такі основні результати. Проведено детальний огляд предметної області та існуючих промислових SIEM-систем (Splunk Enterprise Security, Elastic Security, Wazuh, Datadog Security Monitoring) і виявлено їх ключові недоліки: висока вартість, vendor lock-in, обмежена інтерпретованість висновків та відсутність вбудованої природно-

мовної діагностики з підкріпленням контекстом схожих історичних випадків. На основі порівняльного аналізу обґрунтовано вибір алгоритму виявлення аномалій Isolation Forest [12], трьох класичних методів supervised класифікації (Logistic Regression, Random Forest та XGBoost з технікою SMOTE [17; 18]), а для шару RAG — векторного сховища ChromaDB [22] та моделі ембедингів MiniLM-L6-v2 [21].

Розроблено серверний програмний комплекс на основі трирівневої архітектури, що поєднує детермінований рушій правил R1–R4, статистичний ML-шар детекції аномалій з індивідуальним профілем активності та шар LLM-діагностики з RAG; технологічний стек включає Python 3.11, FastAPI [26], Pydantic v2, SQLAlchemy 2.0, PostgreSQL, ChromaDB, sentence-transformers, Anthropic API, AWS Bedrock і Streamlit. Реалізовано чотири взаємозамінні рушії AI-діагностики (Bedrock baseline, Claude + RAG, LoRA local student, function-calling agent), що повертають однотипну структуровану відповідь у форматі Status / Root Cause / Recommendations і забезпечують оператору гнучкість вибору між якістю діагностики, вартістю виклику та рівнем приватності даних. Додатково реалізовано шар захисту від prompt-injection атак за принципом глибокого захисту з п’ятьма механізмами (regex-детектор, UUID-обгортання, детектор кодувань, інструкції системного промпта, обмеження прав агента), що на корпусі з 27 атак забезпечив 100 % покриття виявлення [27].

Виконано дистиляцію знань з teacher-моделі (Claude Haiku 4.5) у локальну student-модель (Qwen2.5-1.5B + LoRA-адаптер) методом QLoRA [24; 25] на безкоштовному GPU NVIDIA T4: student-модель відтворила 85 % якості teacher-моделі на двох незалежних A/B-прогонах із розбіжністю лише 0,002 в overall-метриці, що є найсильнішим підтвердженням узагальнюваності результату. Компоненти системи розгорнуто у хмарних середовищах Railway (основний бекенд і дашборд) та Azure Container Apps (LoRA-сервіс)

з трирівневим контролем витрат, причому витрати на виклики LLM API під час експериментальної фази склали приблизно 1,37 USD, а маргінальна вартість виклику локального рушія у штатному режимі дорівнює нулю. Нарешті, розроблено інтерактивний дашборд оператора на основі Streamlit з чотирма основними розділами та лабораторією AI-аналізу й складено інструкцію користувача з чотирма основними сценаріями експлуатації.

Можливі галузі використання розробленої системи. Розроблений програмний продукт може бути використаний як готове рішення для моніторингу безпеки інфраструктури малих та середніх організацій, що не мають фінансових ресурсів для впровадження пропрієтарних SIEM-систем. Локальний рушій інференсу на базі дистильованої моделі забезпечує можливість роботи в умовах обмежень на передавання даних за межі периметру організації, що є істотним для державних установ, фінансових організацій та медичних систем, які підпадають під регуляторні обмеження щодо обробки персональних даних [2; 3; 29]. Архітектура з трьома незалежними рівнями детекції (правила + ML + LLM з RAG) дозволяє інтегрувати систему як додатковий рівень у наявну інфраструктуру моніторингу, не замінюючи її повністю.

Рекомендації до застосування. Перед промисловою експлуатацією рекомендовано виконати адаптацію базового профілю активності (baseline) до конкретних характеристик інфраструктури організації за період не менше 30 днів, налаштувати пороги правил R1–R4 під типове навантаження конкретного середовища, ротувати API-ключі регулярно (не рідше ніж раз на квартал) відповідно до зведу практик ДСТУ ISO/IEC 27002:2015 [29]. Для роботи у режимах з підвищеними вимогами до приватності даних рекомендовано увімкнення лише локального рушія LoRA з вимкненням зовнішніх викликів Anthropic API та AWS Bedrock. У режимі підвищеного навантаження доцільно горизонтально масштабувати бекенд через додаткові репліки Railway, що не потребує змін у коді.

Усі поставлені у вступі задачі дипломного проєкту виконано в повному обсязі. Перспективними напрямками подальших досліджень є: розширення корпусу історичних інцидентів для покращення RAG (поточні 500 синтетичних описів — нижня межа для функціональної демонстрації); тестування function-calling агента на сценаріях змінної глибини запитів, де його переваги можуть бути продемонстровані повніше; ablation на розмірі тренувального корпусу для дистиляції (поточні 200 пар можна довести до 1000–2000 за помірних додаткових витрат); дослідження альтернативних базових моделей для локального інференсу (Phi-3-mini, TinyLlama-1.1B, Llama-3.2-1B). Окремим напрямком є інтеграція розробленої системи з еталонними бенчмарками з виявлення вторгнень — крім використаних HDFS_v1 [7; 19] та CICIDS2017 [8], доцільно розглянути сучасніший CICIDS2018 та UNSW-NB15 для перевірки узагальнюваності класифікаторів на нові класи атак.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] *Інформаційна та кібербезпека: соціотехнічний аспект : підручник* / В. Л. Бурячок [та ін.] ; за ред. В. Б. Толубко. — К. : ДУТ, 2015. — 288 с.

[2] *Про основні засади забезпечення кібербезпеки України : Закон України від 05.10.2017 № 2163-VIII*. — 2017. — Відомості Верховної Ради України. 2017. № 45. Ст. 403.

[3] *ДСТУ ISO/IEC 27001:2015. Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. Вимоги (ISO/IEC 27001:2013, IDT)* / ДП «УкрНДНЦ». — К., 2016. — 28 с. — Введ. 2017-01-01.

[4] *ДСТУ ISO/IEC 27035-1:2018. Інформаційні технології. Методи захисту. Керування інцидентами інформаційної безпеки. Частина 1. Принципи керування інцидентами (ISO/IEC 27035-1:2016, IDT)* / ДП «УкрНДНЦ». — К., 2018. — 24 с. — Введ. 2019-01-01 (наказ ДП «УкрНДНЦ» від 10.12.2018).

[5] *Положення про виконання дипломних проєктів освітнього ступеня «бакалавр» на кафедрі обчислювальної техніки ФІОТ* / НТУУ «Київський політехнічний інститут імені Ігоря Сікорського». — К., 2023. — 61 с.

[6] *Лужецький В. А. Інформаційна безпека : навчальний посібник* / В. А. Лужецький, О. П. Войтович, А. В. Дудатьєв. — Вінниця : УНІВЕРСУМ-Вінниця, 2009. — 240 с.

[7] *Detecting Large-Scale System Problems by Mining Console Logs* / W. Xu [та ін.] // Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP'09). — New York : ACM, 2009. — С. 117—132. — DOI: 10.1145/1629575.1629587.

[8] *Sharafaldin I. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization* / I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani // Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP'18). — SciTePress, 2018. — С. 108—116. — DOI: 10.5220/0006639801080116.

[9] *Корченко О. Г. Системи захисту інформації : монографія* / О. Г. Корченко. — К. : НАУ, 2004. — 264 с.

[10] *Драгунцов Р. О. Потенційні відволікаючі атаки на операційні центри безпеки та SIEM системи* / Р. О. Драгунцов, Д. М. Рабчун // Кібербезпека: освіта, наука, техніка. — 2021. — Т. 2, № 14. — С. 6—16. — DOI: 10.28925/2663-4023.2021.14.614.

[11] *Виявлення аномалій в телекомунікаційному трафіку статистичними методами* / Т. А. Радівілова [та ін.] // Кібербезпека: освіта, наука, техніка. — 2021. — Т. 3, № 11. — С. 183—194. — DOI: 10.28925/2663-4023.2021.11.183194.

[12] *Liu F. T. Isolation Forest* / F. T. Liu, K. M. Ting, Z.-H. Zhou // Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM'08). — Washington, DC : IEEE Computer Society, 2008. — С. 413—422. — DOI: 10.1109/ICDM.2008.17.

[13] *Estimating the Support of a High-Dimensional Distribution* / B. Schölkopf [та ін.] // Neural Computation. — 2001. — Т. 13, № 7. — С. 1443—1471. — DOI: 10.1162/089976601750264965.

[14] *LOF: Identifying Density-Based Local Outliers* / M. M. Breunig [та ін.] // Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. — New York : ACM, 2000. — С. 93—104. — DOI: 10.1145/342009.335388.

[15] *Грайворонський М. В. Безпека інформаційно-комунікаційних систем : підручник* / М. В. Грайворонський, О. М. Новіков ; за ред. М. З. Згуровський. — К. : BHV, 2009. — 608 с.

[16] *Субач І. Ю. Модель виявлення кібернетичних атак на інформаційно-телекомунікаційні системи на основі описання аномалій їх роботи зваженими нечіткими правилами* / І. Ю. Субач, В. М. Фесьоха // Information Technology and Security. — 2017. — Т. 5, № 2. — С. 145—152. — DOI: 10.20535/2411-1031.2017.5.2.136984.

[17] *Chen T. XGBoost: A Scalable Tree Boosting System* / T. Chen, C. Guestrin // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. — New York : ACM, 2016. — С. 785—794. — DOI: 10.1145/2939672.2939785.

[18] *SMOTE: Synthetic Minority Over-sampling Technique* / N. V. Chawla [та ін.] // Journal of Artificial Intelligence Research. — 2002. — Т. 16. — С. 321—357. — DOI: 10.1613/jair.953.

[19] *Loghub: A Large Collection of System Log Datasets for AI-Driven Log Analytics* / J. Zhu [та ін.] // Proceedings of the 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE'23). — Florence : IEEE, 2023. — С. 355—366. — DOI: 10.1109/ISSRE59848.2023.00071.

[20] *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks* / P. Lewis [та ін.] // Advances in Neural Information Processing Systems 33 (NeurIPS'20). — Curran Associates, 2020. — С. 9459—9474.

[21] *Reimers N. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks* / N. Reimers, I. Gurevych // Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP'19). — Stroudsburg : ACL, 2019. — С. 3982—3992. — DOI: 10.18653/v1/D19-1410.

[22] *Chroma*. *ChromaDB documentation* / Chroma. — Режим доступу до ресурсу: https://docs.trychroma.com/ (дата зверн. 20.05.2026).

[23] *Hinton G. Distilling the Knowledge in a Neural Network* / G. Hinton, O. Vinyals, J. Dean. — 2015. — arXiv preprint arXiv:1503.02531.

[24] *LoRA: Low-Rank Adaptation of Large Language Models* / E. J. Hu [та ін.] // Proceedings of the International Conference on Learning Representations (ICLR'22). — 2022.

[25] *QLoRA: Efficient Finetuning of Quantized LLMs* / T. Dettmers [та ін.] // Advances in Neural Information Processing Systems 36 (NeurIPS'23). — Curran Associates, 2023. — С. 10088—10115.

[26] *Щербина І. С. Вплив технології Docker на архітектуру мікросервісів* / І. С. Щербина, Д. В. Явор // Наукові записки Державного університету інформаційно-комунікаційних технологій. — 2023. — № 1. — С. 65—70.

[27] *Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection* / K. Greshake [та ін.] // Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec'23). — New York : ACM, 2023. — С. 79—90. — DOI: 10.1145/3605764.3623985.

[28] *Аналіз процесів використання Docker для побудови мікросервісів* / О. А. Андрушко [та ін.] // Науковий вісник НЛТУ України. — 2017. — Т. 27, № 9. — С. 95—98.

[29] *ДСТУ ISO/IEC 27002:2015. Інформаційні технології. Методи захисту. Звід практик щодо заходів інформаційної безпеки (ISO/IEC 27002:2013, IDT)* / ДП «УкрНДНЦ». — К., 2016. — 90 с. — Введ. 2017-01-01.

[30] *ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення* / Держстандарт України. — К., 1995. — 36 с.

[31] *ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання* / ДП «УкрНДНЦ». — К., 2016. — 16 с. — Введ. 2016-07-01.

[32] *Engelen G. Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study* / G. Engelen, V. Rimmer, W. Joosen // Proceedings of the IEEE Symposium on Security and Privacy Workshops (SPW'21). — Piscataway : IEEE, 2021. — С. 7—12. — DOI: 10.1109/SPW53761.2021.00009.

ДОДАТОК А

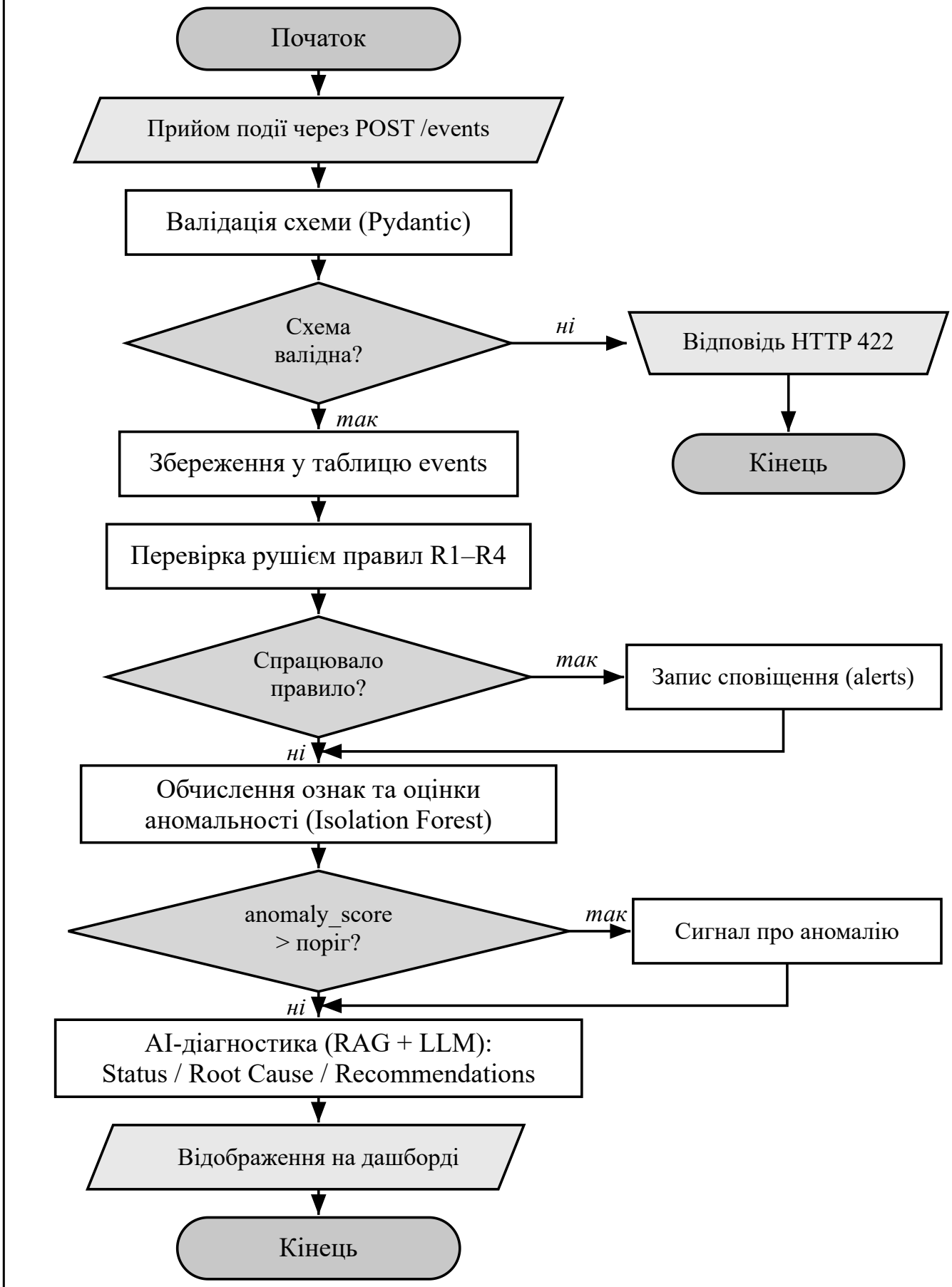
Система збору та аналізу сигналів безпеки комп’ютерної інфраструктури

Блок-схема алгоритму

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ — 2026 р.



					ІАЛЦ.467200.004 Д1	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Боднар А. А.			**Система збору та аналізу сигналів безпеки комп'ютерної інфраструктури**	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Сергієнко А. М.				
Реценз.		Молчанов О. А.				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-23
Н. Контр.		Коренко Д. В.				
Затверд.		Волокита А. М.			Блок-схема алгоритму	

ДОДАТОК Б

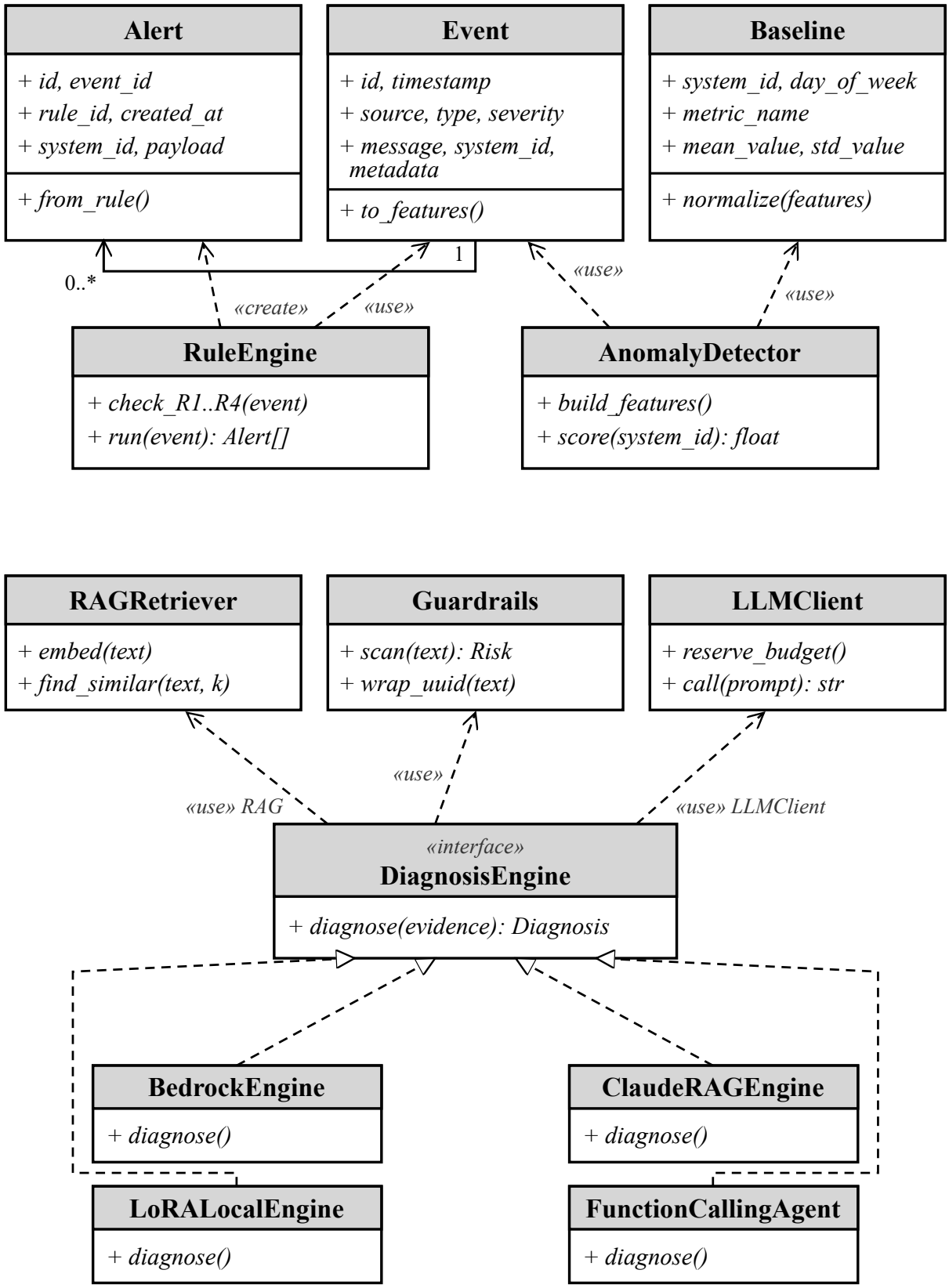
Система збору та аналізу сигналів безпеки комп’ютерної інфраструктури

Схема функціональна

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ — 2026 р.



					ІАЛЦ.467200.005 Д2	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Боднар А. А.			**Система збору та аналізу сигналів безпеки комп'ютерної інфраструктури**	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Сергієнко А. М.				
Реценз.		Молчанов О. А.				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-23
Н. Контр.		Коренко Д. В.				
Затверд.		Волокита А. М.			Схема функціональна	

ДОДАТОК В

Система збору та аналізу сигналів безпеки комп’ютерної інфраструктури

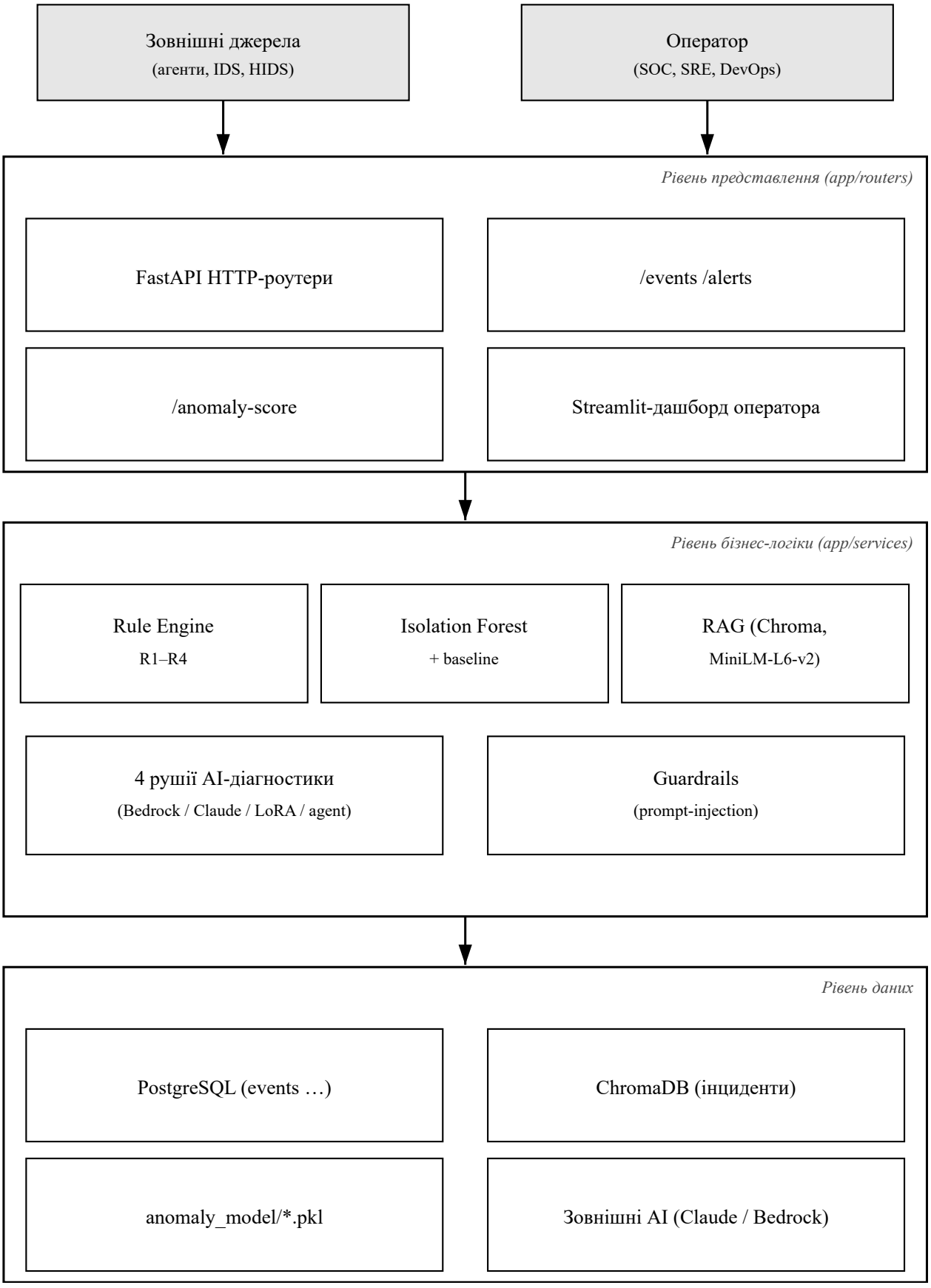
Схема структурна

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ — 2026 р.

Трирівнева архітектура серверного програмного комплексу



					ІАЛЦ.467200.006 Д3	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Боднар А. А.			**Система збору та аналізу сигналів безпеки комп'ютерної інфраструктури** Схема структурна	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Сергієнко А. М.				
Реценз.		Молчанов О. А.				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-23
Н. Контр.		Коренко Д. В.				
Затверд.		Волокита А. М.				

ДОДАТОК Г

Система збору та аналізу сигналів безпеки комп’ютерної інфраструктури

Текст програми

ІАЛЦ.467200.007 Д4

Аркушів 1

Київ — 2026 р.

Повний текст програми розміщено у відкритому репозиторії проєкту за посиланням, наведеним нижче (продубльовано QR-кодом).

https://github.com/4ndruxa/bodnar_diploma_project

					ІАЛЦ.467200.007 Д4			
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив		Боднар А. А.			**Система збору та аналізу сигналів безпеки комп'ютерної інфраструктури** Текст програми	Літ.	Аркуш	Аркушів
Перевірив		Сергієнко А. М.					1	1
Реценз.		Молчанов О. А.				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-23		
Н. Контр.		Коренко Д. В.						
Затверд.		Волокита А. М.						