МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Кваліфікаційна наукова праця
на правах рукопису

ШТАНЬКО ВАДИМ ІГОРОВИЧ

УДК 004.9:004.056

ДИСЕРТАЦІЯ

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ДВОРІВНЕВОЇ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ АНАЛІЗУ МЕРЕЖЕВИХ АТАК

122 «Комп'ютерні науки»
(12 – «Інформаційні технології»)

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень.
Використання ідей, результатів, текстів інших авторів мають посилання на відповідне джерело

В.І. Штанько

Науковий керівник
Нікітенко Євгеній Васильович,
кандидат фізико-математичних наук, доцент

Київ 2026

АНОТАЦІЯ

Штанько В. І. Інформаційна технологія дворівневої інтелектуальної системи аналізу мережевих атак. - Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії з технічних наук за спеціальністю 122 «Комп'ютерні науки». Національний університет біоресурсів і природокористування України, Київ, 2026 рік.

У дисертації досліджено методи та моделі аналізу мережевого трафіку, алгоритми машинного навчання та підходи до побудови дворівневих систем виявлення вторгнень у контексті забезпечення адаптивного кіберзахисту інформаційно-комунікаційних систем. Окреслені теоретичні засади формують методологічну основу для створення інформаційної технології дворівневої інтелектуальної системи аналізу мережевих атак, що забезпечує автоматизоване виявлення, класифікацію та інтерпретацію атипової мережевої активності у віртуалізованому середовищі. Актуальність дослідження зумовлена зростанням інтенсивності та складності кібератак, поширенням гібридних загроз, а також необхідністю переходу від сигнатурного до поведінкового аналізу мережевого трафіку із використанням методів машинного навчання.

Розвиток хмарних технологій, розподілених обчислювальних середовищ і сервіс-орієнтованих архітектур обумовлює підвищення вимог до систем виявлення вторгнень щодо масштабованості, адаптивності та здатності функціонувати в умовах змінного статистичного профілю трафіку. Традиційні сигнатурні підходи демонструють обмежену ефективність у разі появи нових або модифікованих сценаріїв атак. У зв'язку з цим обґрунтовано доцільність використання ймовірнісних та ансамблевих моделей машинного навчання як складових ієрархічної системи прийняття рішень.

На основі публічного набору даних USB-IDS-1 та власноруч сформованих наборів трафіку, отриманих в ізольованому віртуальному середовищі на базі GNS3 та VirtualBox, створено емпіричну базу дослідження обсягом понад 6 млн мережевих пакетів, агрегованих у потокове представлення з використанням віконної сегментації. Це дозволило провести комплексний аналіз статистичних

характеристик трафіку, сформувати навчальні та тестові вибірки для різних сценаріїв функціонування системи та дослідити вплив зсуву домену (domain shift) на якість класифікації. Встановлено, що зміна розподілів ознак істотно впливає на поведінку моделей, що підтверджує необхідність формалізованого врахування цього явища в процедурі прийняття рішень.

Встановлено, що використання на першому рівні ієрархії наївного баєсівського класифікатора як бінарного фільтра забезпечує ефективне відокремлення нормативного трафіку від атак із мінімальними обчислювальними витратами. На другому рівні застосування ансамблевих методів, зокрема випадкового лісу, дозволяє здійснювати детальну багатокласову атрибуцію типів атак. У режимі узгодженого домену середнє значення accuracy для бінарної класифікації досягає 0,97, а для багатокласової – 0,89, що свідчить про високу здатність системи до розпізнавання типів вторгнень. У разі зміни статистичного профілю трафіку спостерігається закономірна деградація метрик, що підтверджує чутливість моделей до зсуву домену та необхідність адаптивного налаштування порогів прийняття рішень.

Обґрунтовано математичну модель дворівневої системи класифікації як задачу мінімізації очікуваного баєсівського ризику з урахуванням асиметрії вартості помилок першого та другого роду. Аналітично виведено порогові умови прийняття рішення для кожного рівня ієрархії. Запропоновано механізм адаптивного коригування порогів на основі зваженої дивергенції Кульбака–Лейблера, що дозволяє кількісно оцінювати ступінь статистичної розбіжності між поточним та еталонним трафіком і відповідно регулювати чутливість системи без повного перенавчання моделей.

Розроблено архітектуру інформаційної технології, яка реалізує замкнений цикл «генерація – детерміноване маркування – потокова класифікація» у віртуалізованому середовищі. Використання детермінованого маркування пакетів на етапі емуляції забезпечує формування верифікованих наборів даних для навчання та тестування моделей, що підвищує достовірність експериментальних результатів. Перехід від статичних інтегральних показників до синхронного віконного аналізу метрик надійності дав змогу оцінювати

стабільність функціонування системи у часовому розрізі та виявляти деградацію якості класифікації в динамічних умовах.

Одержані результати реалізовано у вигляді програмної системи аналізу мережевого трафіку з керованою політикою прийняття рішень, що передбачає механізм обробки невизначених станів («Unknown») та процедуру відмови від рішення у випадках підвищеної статистичної невизначеності. Запропонована модульна архітектура забезпечує можливість інтеграції з наявними засобами моніторингу мережевої безпеки та підтримує масштабування з урахуванням появи нових типів кіберзагроз. Реалізований підхід створює передумови для впровадження адаптивних систем виявлення вторгнень і підвищення стійкості інформаційно-комунікаційних систем до сучасних кібератак.

Ключові слова: інформаційна технологія, інформаційна система, машинне навчання, інтелектуальний аналіз даних, моделі прийняття рішень, класифікація даних, мережі, мережевий трафік, віртуальне середовище, система виявлення вторгнень, мережева атака, DDoS-атака, Python.

ABSTRACT

Shtanko V. I. Information Technology for a Two-Level Intelligent Network Attack Analysis System. The qualification scientific work on the rights of the manuscript.

The dissertation on acquisition of Doctor of Philosophy scientific degree in technical sciences in the specialty 122 «Computer sciences». National University of Life and Environmental Sciences of Ukraine, Kyiv, 2026.

The dissertation investigates methods and models for network traffic analysis, machine learning algorithms, and approaches to building two-level intrusion detection systems in the context of ensuring adaptive cybersecurity of information and communication systems. The outlined theoretical foundations form the methodological basis for creating an information technology of a two-level intelligent system for network attack analysis, which provides automated detection, classification, and interpretation of anomalous network activity in a virtualized environment. The relevance of the study is conditioned by the growing intensity and complexity of cyberattacks, the spread of hybrid threats, and the need to shift from signature-based to behavioral analysis of network traffic using machine learning methods.

The development of cloud technologies, distributed computing environments, and service-oriented architectures increases the requirements for intrusion detection systems in terms of scalability, adaptability, and the ability to operate under dynamically changing traffic profiles. Traditional signature-based approaches demonstrate limited effectiveness in the case of new or modified attack scenarios. In this regard, the feasibility of using probabilistic and ensemble machine learning models as components of a hierarchical decision-making system is substantiated.

Based on the public USB-IDS-1 dataset and custom traffic datasets collected in an isolated virtual environment using GNS3 and VirtualBox, an empirical research base was formed, comprising more than 6 million network packets aggregated into flow-based representations with window segmentation. This enabled a comprehensive analysis of traffic statistical characteristics, the formation of training and test datasets for various operational scenarios, and the study of the impact of domain shift on classification performance. It was established that changes in feature distributions

significantly affect model behavior, confirming the necessity of formally incorporating this phenomenon into the decision-making procedure.

It was found that using a Naive Bayes classifier as a binary filter at the first level of the hierarchy ensures effective separation of normal traffic from attacks with minimal computational cost. At the second level, the application of ensemble methods, particularly Random Forest, enables detailed multiclass attribution of attack types. In the in-domain setting, the average accuracy reaches 0.97 for binary classification and 0.89 for multiclass attribution, indicating a high capability of the system to identify intrusion types. When the statistical profile of traffic changes, a consistent degradation of metrics is observed, confirming model sensitivity to domain shift and the necessity of adaptive threshold adjustment.

A mathematical model of the two-level classification system is substantiated as a problem of minimizing expected Bayesian risk with consideration of asymmetric costs of type I and type II errors. Threshold decision rules for each hierarchical level are derived analytically. An adaptive threshold adjustment mechanism based on weighted Kullback–Leibler divergence is proposed, allowing quantitative estimation of statistical divergence between current and reference traffic and enabling sensitivity regulation without full model retraining.

An information technology architecture has been developed that implements a closed-loop cycle of “generation – deterministic labeling – flow-based classification” in a virtualized environment. The use of deterministic packet labeling at the emulation stage ensures the formation of verified datasets for model training and testing, thereby increasing the reliability of experimental results. The transition from static integral metrics to synchronous window-based reliability analysis enables the assessment of system stability over time and the detection of classification performance degradation under dynamic operating conditions.

The obtained results are implemented in the form of an program system for network traffic analysis with a controlled decision-making policy, including a mechanism for handling uncertain states (“Unknown”) and a rejection option in cases of increased statistical uncertainty. The proposed modular architecture ensures integration with existing network security monitoring tools and supports scalability to

accommodate new types of cyber threats. The implemented approach creates the prerequisites for deploying adaptive intrusion detection systems and enhancing the resilience of information and communication systems against modern cyberattacks.

Keywords: information technology, information system, machine learning, intelligent data analysis, decision-making models, data classification, networks, network traffic, virtual environment, intrusion detection system, network attack, DDoS attack, Python.

СПИСОК ОПУБЛІКОВАНИХ ПРАЦЬ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Стаття у науковому виданні, включеному до міжнародних наукометричних баз даних Scopus та/або Web of Science Core Collection

1. Konyrbaev N., Nikitenko Ye., **Shtanko V.**, Lakhno V., Baishemirov Z., Ibadulla S., Galymzhankyzy A., Myrzabek E. Evaluation and Optimization of the Naive Bayes Algorithm for Intrusion Detection Systems Using the USB-IDS-1 Dataset. Eastern-European Journal of Enterprise Technologies. 2024. Vol. 6. No. 2 (132). P. 74–82. (*Konyrbaev N. сформулював концепцію дослідження та визначив загальну постановку задачі оцінювання ефективності алгоритму Naive Bayes в IDS. Nikitenko Y. визначив наукову гіпотезу щодо залежності ефективності моделі від кількості записів і кількості класів та забезпечив методологічне керівництво дослідженням. Shtanko V. реалізував модель Gaussian Naive Bayes у середовищі Python, виконав підготовку та оброблення даних USB-IDS-1, сформував експериментальний протокол із двома групами розрахунків (залежність від обсягу даних та від кількості класів), здійснив обчислення accuracy та precision, провів регресійний аналіз отриманих результатів і сформулював висновки щодо обмежень алгоритму у багатокласовому режимі. Lakhno V. здійснив інтерпретацію результатів дослідження в контексті практичного застосування систем виявлення вторгнень. Myrzabek E визначив структуру експериментального дизайну та організацію ітераційних обчислень. Ibadulla S. забезпечив формалізацію структури вхідних даних та підготовку наборів для експериментального аналізу. Galymzhankyzy A. виконала статистичну обробку експериментальних результатів та підготовку графічної інтерпретації залежностей. Baishemirov Z. здійснив узагальнення результатів дослідження та підготовку матеріалів до публікації.*).

Стаття у науковому виданні, включеному до категорії «Б» Переліку наукових фахових видань України

2. **Штанько В.**, Нікітенко Є. Проєктування та реалізація віртуального середовища для аналізу мережевого трафіку. Наука і техніка сьогодні. 2025.

№ 7 (48). С. 2028–2045. (*Штанько В. І. спроєктував архітектуру ізольованого віртуального середовища на базі GNS3 та VirtualBox, реалізував мережеву топологію з використанням маршрутизатора Cisco 2600, налаштував IP-адресацію, маршрутизацію та сегментацію підмереж, розгорнув вузол-«зловмисник» на Kali Linux та вузли-«жертви» на Debian Linux, забезпечив повну ізоляцію віртуальної мережі від зовнішнього середовища, виконав верифікацію топології за допомогою ping, traceroute та аналізу ICMP-пакетів у Wireshark, а також здійснив експериментальну перевірку коректності маршрутизації та мережевої ізоляції. Нікітенко Є. В. сформулював концептуальні засади побудови ізольованого дослідницького середовища для аналізу мережевих атак, визначив методологічні вимоги до контрольованості та відтворюваності експериментальної платформи, а також здійснив науковий супровід і редакційне опрацювання матеріалів*).

Тези наукових доповідей

3. **Штанько В. І.**, Нікітенко Є. В. Актуальні методи побудови систем виявлення вторгнень. Теоретичні та прикладні аспекти розробки комп'ютерних систем '2023': V Всеукраїнська науково-практична конференція студентів і аспірантів, м. Київ, 26 квітня 2023 року: тези доповіді. Київ, 2023. С. 180–181. (*Штанько В. І. здійснив огляд наявних методів побудови систем виявлення вторгнень, виконав аналіз класифікації IDS за методами розгортання та способами виявлення атак, узагальнив переваги й обмеження сигнатурних та аномалійних підходів, а також сформулював висновки щодо доцільності використання сучасних методів у системах захисту інформації. Нікітенко Є. В. визначив концептуальні засади дослідження та здійснив наукове керівництво підготовкою тез.*).

4. Штанько В. І. Порівняльний аналіз навчальних наборів даних для машинного навчання систем виявлення вторгнень. Інформаційні технології: економіка, техніка, освіта '2023': XIV Міжнародна науково-практична конференція молодих вчених, м. Київ, 26–27 жовтня 2023 року: тези доповіді. Київ, 2023. С. 244–245.

5. Штанько В. І. Використання аналізаторів трафіку для побудови систем виявлення вторгнень. Глобальні та регіональні проблеми інформатизації в суспільстві і природокористуванні ‘2024’: XII Міжнародна науково-практична конференція, м. Київ, 21–22 листопада 2024 року: тези доповіді. Київ, 2024. С. 97–99.

6. Штанько В. І. Швидкодія моделі Naive Bayes з відбором ознак для набору USB-IDS-1. Теоретичні та прикладні аспекти розробки комп’ютерних систем ‘2025’: VII Всеукраїнська науково-практична конференція студентів і аспірантів, м. Київ, 24 квітня 2025 року: тези доповіді. Київ, 2025. С. 375–376.

7. Vadym Shtanko. Usage of GNS3 for cybersecurity research. Achievements of Science and Education in the Modern World. Achievements of Science and Education in the Modern World: 2nd International Scientific Conference, Birmingham, United Kingdom, 14 June 2025: Conference Paper. Birmingham, United Kingdom, 2025. P. 128–130.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. **IDS** – *Intrusion Detection System* – система виявлення вторгнень.
2. **IPS** – *Intrusion Prevention System* – система запобігання вторгненням.
3. **DDoS** – *Distributed Denial of Service* – розподілена атака типу «відмова в обслуговуванні».
4. **DoS** – *Denial of Service* – атака типу «відмова в обслуговуванні».
5. **NB** – *Naive Bayes* – наївний баєсівський класифікатор.
6. **RF** – *Random Forest* – метод випадкового лісу.
7. **ML** – *Machine Learning* – машинне навчання.
8. **AI** – *Artificial Intelligence* – штучний інтелект.
9. **DL** – *Deep Learning* – глибоке навчання.
10. **SVM** – *Support Vector Machine* – метод опорних векторів.
11. **CNN** – *Convolutional Neural Network* – згорткова нейронна мережа.
12. **RNN** – *Recurrent Neural Network* – рекурентна нейронна мережа.
13. **TP**, **FP**, **TN**, **FN** – *True Positive, False Positive, True Negative, False Negative* – метрики правильних і хибних класифікацій.
14. **FPR**, **TPR** – *False Positive Rate, True Positive Rate* – частота хибних і правильних спрацьовувань.
15. **OSI** – *Open Systems Interconnection* – еталонна семирівнева модель взаємодії відкритих систем.
16. **TCP** – *Transmission Control Protocol* – протокол керування передачею.
17. **UDP** – *User Datagram Protocol* – протокол дейтаграм користувача.
18. **HTTP** – *HyperText Transfer Protocol* – протокол передачі гіпертексту.
19. **CSV** – *Comma-Separated Values* – формат табличних даних із розділенням комами.
20. **GNS3** – *Graphical Network Simulator 3* – графічний мережевий емулятор.
21. **VM** – *Virtual Machine* – віртуальна машина.

ВСТУП

Актуальність теми дослідження. У сучасному суспільстві з високим рівнем цифровізації інформація є стратегічним ресурсом, що визначає ефективність функціонування економічних, виробничих, наукових та державних систем. Розвиток інформаційних технологій, глобальних комунікацій і мережевої взаємодії призвів до суттєвого зростання обсягів переданих і збережених даних, а також до ускладнення їхніх каналів оброблення. Разом із тим, збільшилася кількість загроз, пов’язаних із несанкціонованим доступом, порушенням цілісності та відмовою в обслуговуванні інформаційних ресурсів.

Сучасні кібератаки, зокрема розподілені атаки типу DDoS, стали високотехнологічними, масштабними та здатними виводити з ладу критично важливі об’єкти інформатизації. Їхня складність, мінливість і автоматизація роблять традиційні системи виявлення вторгнень недостатньо ефективними. В умовах постійного зростання обсягів трафіку, гібридних загроз і швидкої еволюції шкідливих методів виникає потреба у впровадженні інтелектуальних систем виявлення вторгнень (IDS), здатних адаптивно реагувати на нові типи атак у реальному часі.

Одним із напрямів розвитку IDS є використання методів штучного інтелекту (AI) та машинного навчання (ML), які забезпечують автоматизований аналіз мережевого трафіку, класифікацію аномалій і самонавчання системи на основі накопичених даних. Однак висока точність класифікації, досягнута в контрольованих умовах, не гарантує стабільної та надійної роботи в реальному середовищі. Системи виявлення загроз часто змушені ухвалювати рішення за неповної або нестабільної інформації [107], зокрема у випадках змін у поведінці трафіку або появи нетипових ознак. У таких ситуаціях важливою властивістю є здатність моделі фіксувати невизначеність або утримуватись від класифікації за умов низької впевненості [122]. Хибна класифікація потенційної атаки як легітимного трафіку є більш ризикованою, ніж сигнал про невизначеність. У цьому контексті ключовим завданням є розроблення інформаційної технології ухвалення рішень, здатної працювати в умовах відкритої множини ситуацій і забезпечувати контрольований рівень довіри до результатів класифікації.

Однак впровадження таких рішень потребує комплексного підходу – розроблення інформаційної технології дворівневої інтелектуальної системи аналізу мережевих атак, здатної поєднувати алгоритмічну адаптивність, реалістичне моделювання загроз та діагностику подій на різних рівнях моделі OSI.

Таким чином, розроблення універсального інтелектуального середовища, яке дозволить моделювати атаки, збирати й аналізувати трафік, а також здійснювати автоматизовану класифікацію подій, є актуальним науковим і практичним завданням.

Мета роботи – підвищення ефективності виявлення та класифікації мережевих атак в умовах варіативної зміни характеристик трафіку шляхом розробки інформаційної технології дворівневої інтелектуальної системи аналізу мережевих атак на основі ієрархії моделей машинного навчання.

Для досягнення мети поставлено та вирішено такі завдання:

1. Провести аналіз релевантних методів та моделей машинного навчання в системах виявлення вторгнень, виявити їх обмеження щодо зміни розподілу ознак мережевого трафіку (domain shift) та обґрунтувати доцільність використання ансамблевих та ймовірнісних методів та моделей.

2. Розробити модель прийняття рішень для дворівневої системи класифікації, яка базується на мінімізації очікуваного ризику та включає аналітичне виведення порогів спрацювання залежно від рівня інформаційної розбіжності поточного та еталонного трафіку.

3. Розробити архітектуру інформаційної технології, яка поєднує бінарну фільтрацію та поглиблену багатокласову класифікацію із механізмом опрацювання невизначених станів для зменшення навантаження на систему та підвищення точності розпізнавання вторгнень.

4. Створити інструментальні засоби та ізольоване віртуальне середовище на базі GNS3 та VirtualBox для безпечної емуляції складних сценаріїв атак та провести експериментальні дослідження розробленої системи на власних та публічних наборах даних, й виконати порівняльний аналіз ефективності

класифікації, а також підтвердити адекватність запропонованої моделі до умов зсуву домену.

Об’єкт дослідження – процеси моніторингу, виявлення та класифікації атипової мережевої активності в інформаційно-комунікаційних системах.

Предмет дослідження – методи та моделі класифікації трафіку на основі баєсівського ризику, алгоритми налаштування порогів прийняття рішень, архітектура дворівневої ієрархічної системи аналізу мережевих атак та програмні засоби верифікації результатів у віртуалізованому середовищі.

Методи дослідження.

У дисертації використано комплекс теоретичних, математичних та емпіричних методів дослідження. А саме аналітичні методи застосовано для системного аналізу релевантних методів та моделей синтезу систем виявлення вторгнень (IDS), а також для порівняльного аналізу наборів даних, як от KDD’99, CIC-IDS2017, USB-IDS-1 тощо для обґрунтування необхідності створення верифікованих власних наборів даних. Теоретичні методи використано для розв’язання задачі виявлення мережевих атак як процесу прийняття рішень в умовах асиметрії ризиків. Методи системного аналізу та проєктування архітектури IDS застосовано для обґрунтування вибору дворівневої моделі класифікації на основі наївного баєсівського класифікатора та випадкового лісу та розробки інформаційної технології із замкненим циклом «генерація–маркування–класифікація». Математичні методи, зокрема теорія ймовірностей та баєсівський аналіз, дивергенція Кульбака-Лейблера для якої розроблено математичний апарат для кількісної оцінки статистичних розбіжностей у мережевому трафіку та автоматичного корегування чутливості системи. Емпіричні методи, що включали комп’ютерне моделювання в ізольованому віртуальному середовищі GNS3, VirtualBox, застосовано для безпечної емуляції сценаріїв атак. Методи об'єктно-орієнтованого програмування використано для програмної реалізації модулів FlowCapture та тренувальних компонентів на мові Python. Методи експериментального аналізу та статистична обробка результатів дозволили перевірити адекватність запропонованих моделей.

Наукова новизна одержаних результатів полягає у наступному:

Уперше розроблено інформаційну технологію аналізу мережевих атак у віртуалізованому середовищі, яка, на відміну від наявних, реалізує замкнений цикл «генерація – маркування – потокова класифікація» та використовує детерміноване маркування пакетів на етапі емуляції, що забезпечує формування верифікованих наборів даних для навчання моделей без необхідності евристичної розмітки;

Удосконалено метод виявлення вторгнень шляхом синтезу моделі налаштування порогів класифікації, яка, на відміну від чинних, базується на критерії мінімізації баєсівського ризику та враховує кількісну оцінку стохастичного зсуву домену на основі зваженої дивергенції Кульбака-Лейблера, що дозволяє забезпечити автоматичне коригування чутливості системи до змін параметрів мережевого трафіку та мінімізувати вартість помилкових рішень в умовах варіативної зміни мережевого середовища;

Набув подальшого розвитку метод оцінювання ефективності систем виявлення та класифікації мережевих атак шляхом переходу від статичних метрик до синхронного віконного аналізу показників надійності, що дає змогу виявляти деградацію якості класифікації при зміні статистичного профілю трафіку;

Набув подальшого розвитку метод проєктування архітектури систем виявлення вторгнень шляхом впровадження дворівневої ієрархічної структури класифікаторів на основі наївного баєсівського класифікатора та випадкового лісу із впровадженням класу невизначеності «Unknown» та механізму відмови від рішення, що забезпечує узгодженість між обчислювальною ефективністю первинної фільтрації трафіку та точністю атрибуції типу атаки.

Практичне значення отриманих результатів полягає у тому, що запропоновані моделі та методи прийняття рішень, зокрема дворівнева ієрархічна модель класифікації та механізм адаптивного налаштування порогів на основі оцінки зсуву домену, реалізовано в експериментальній програмній системі аналізу мережевого трафіку. Розроблена система інтегрує модель бінарної фільтрації та багатокласової атрибуції з механізмом відмови від

рішення, що забезпечує керовану політику прийняття рішень у задачах виявлення мережевих атак. Реалізований алгоритм адаптивного коригування порогів дозволяє зберігати стабільність функціонування системи в умовах зміни статистичних характеристик трафіку без необхідності повного перенавчання моделей. Практичну реалізацію виконано у вигляді програмного засобу мовою Python, який включає модуль перехоплення та агрегації потоків (FlowCapture), а також об’єктно-орієнтовані компоненти навчання та застосування моделей (TrainerBayes, TrainerRandomForest). Система автоматизує повний цикл оброблення даних – від захоплення сирого трафіку до дворівневої класифікації в режимі, наближеному до синхронного. Для експериментальної перевірки моделей створено ізольоване віртуалізоване середовище на базі GNS3 та VirtualBox, що дозволяє безпечно моделювати сценарії атак, формувати марковані набори потокових даних та здійснювати верифікацію алгоритмів IDS/IPS у контрольованих умовах. Запропонована модульна архітектура системи придатна до інтеграції з іншими засобами моніторингу мережевої безпеки та масштабування під нові типи кіберзагроз.

Результати дисертації було впроваджено у навчальний процес кафедри комп'ютерних систем, мереж та кібербезпеки Національного університету біоресурсів і природокористування України, про що свідчить акт про впровадження результатів дисертації (додаток А).

Особистий внесок здобувача. Усі наукові результати, положення та висновки одержані автором самостійно. Розроблено архітектуру системи, алгоритмічну схему дворівневої класифікації, проведено моделювання атак у віртуальному середовищі, реалізовано програмний модуль аналізу трафіку, здійснено експериментальні дослідження та оцінку ефективності класифікаторів.

Апробація результатів дослідження

Результати дисертаційного дослідження були представлені на міжнародних і всеукраїнських науково-технічних конференціях.

- V Всеукраїнська науково-практична конференція студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп’ютерних систем ‘2023», 2023, НУБіП України, Київ.

- XIV Міжнародна науково-практична конференція молодих вчених «Інформаційні технології: економіка, техніка, освіта ‘2023», 2023, тези доповіді, НУБіП України, Київ.

- XII Міжнародна науково-практична конференція „Глобальні та регіональні проблеми інформатизації в суспільстві і природокористуванні '2024», 2024, НУБіП України, Київ.

- VII Всеукраїнська науково-практична конференція студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп’ютерних систем ‘2025», 24 квітня 2025 року, НУБіП України, Київ.

- 2nd International Scientific Conference Achievements of Science and Education in the Modern World (Birmingham, United Kingdom, 14 June 2025.

Публікації.

Основні результати дослідження відображено у 7 наукових працях, серед яких статті у фахових виданнях та виданнях, включених до баз Scopus, а також матеріали міжнародних конференцій.

Обсяг і структура дисертації.

Дисертація складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел і додатків. Загальний обсяг становить 183 сторінки, у тому числі 26 рисунків по тексту, 5 таблиць по тексту, список джерел із 126 найменувань на 9 сторінках, 10 додатків на 32 сторінках.

РОЗДІЛ 1
АНАЛІЗ СТАНУ ТА ТЕНДЕНЦІЙ РОЗВИТКУ МЕТОДІВ МАШИННОГО НАВЧАННЯ В СИСТЕМАХ ВИЯВЛЕННЯ ВТОРГНЕНЬ

Системи виявлення вторгнень (Intrusion Detection Systems, IDS) є одними із найважливіших сучасних засобів кібербезпеки, оскільки вони надають можливість автоматично розпізнавати зловмисні дії або атаки як в мережах, так і на окремих комп'ютерах. З огляду на те що кількість та складність кібератак постійно зростає [14], виникає потреба у більш просунутих системах, які використовують штучний інтелект і які здатні ефективно виявляти як відомі, так і нові загрози в реальному масштабі часу. Впродовж останніх років у цій галузі активно застосовуються методи машинного навчання (Machine Learning - ML) і штучного інтелекту (Artificial Intelligence - AI) [34]. Використання цих інструментів дозволяє істотно підвищити адаптивність і точність IDS порівняно з класичними системами, які використовують сигнатури.

IDS поділяються на дві великі групи: сигнатурні виявляють відомі атаки за наперед заданими шаблонами, інша група фокусується на виявленні аномалій, тобто відхилень від нормальної поведінки. Використання методів машинного навчання помітно розширило можливості обох типів IDS. У ранніх роботах 2000-х років широко розглядалися класичні алгоритми машинного навчання: методи на основі правил і дерев рішень (C4.5 [96], CART [125]), наївний класифікатор Баєса [68], метод опорних векторів (Supporting Vector Machine, SVM), Random Forest [121] тощо. Хоча ці алгоритми продемонстрували досить гарні результати при використанні стандартних наборів даних (таких як KDD Cup 99), однозначної відповіді щодо того, який саме алгоритм є найбільш ефективним, немає [35]. Наприклад, у огляді 2019 р. зазначено, що багато робіт, які використовували KDD-99 як навчальний набір даних, не дійшли остаточного висновку щодо переваги якогось із методів класифікації. Крім того, не у всіх роботах брався до уваги критична для практики швидкодія алгоритмів [35]. Отже, класичні підходи кожен мають свої переваги і недоліки: SVM здатний забезпечувати високу точність для задач двійкової класифікації, але досить

вимогливий до обчислень на великих вибірках; Random Forest є стійким до перенавчання та допускає небагато помилок при класифікації завдяки усередненню багатьох дерев [53], однак моделі Random Forest складно інтерпретувати; наївний Баєсівський класифікатор дуже швидкий, проте для нього необхідно, щоб ознаки були незалежними одна від одної, а це знижує його точність на складних мережевих даних [37].

За останнє десятиріччя дослідження здебільшого зосереджуються на сучасних методах штучного інтелекту: глибокому навчанні (Deep Learning, DL) та ансамблевих моделях [34]. Між 2018 та 2023 роками спостерігається поява значної кількості публікацій, які присвячені глибоким нейронним мережам, еволюційним та оптимізаційним алгоритмам в IDS [34]. До найбільш уживаних алгоритмів у цій сфері належать згорткові нейронні мережі (Convolutional Neural Network, CNN [80]), метод опорних векторів, дерева рішень та генетичні алгоритми [34]. Саме ці методи згідно з результатами досліджень показують високі показники виявлення вторгнень [4; 5; 20; 23; 24]. Таке широке застосування глибоких нейронних мереж пояснюється тим, що вони спроможні автоматично виокремлювати шаблони в мережевому трафіку і виявляти складні залежності між ознаками мережевих даних, що дуже важко реалізувати засобами традиційних алгоритмів. Зокрема, для виявлення аномалій дослідники використовували автоенкодери [46], рекурентні нейронні мережі (Recurrent Neural Networks, RNN) для аналізу послідовностей пакетів тощо. Крім того, активно досліджуються ансамблеві й гібридні підходи: поєднання кількох алгоритмів з метою підвищити надійність і точність IDS. До прикладу, метод бустингу застосовується для побудови групи слабких класифікаторів, що спільно формують сильний детектор вторгнень [40]. Популярні алгоритми бустингу, такі як XGBoost та LightGBM показали перевагу над окремими моделями за рахунок усереднення їхніх помилок. Гібридні IDS комбінують сигнатурні та аномалійні підходи або різні рівні аналізу: наприклад, одна підсистема виконує швидку фільтрацію відомих атак за сигнатурами, а інша здійснює більш детальний аналіз, виявляючи аномалії за допомогою ML. Подібна комбінація забезпечує меншу кількість хибнопозитивних спрацьовувань і одночасно виявляти нові

загрози [27]. В дослідженнях описано низку різноманітних архітектур, в яких традиційні алгоритми поєднуються з глибинними нейромережами або евристичними оптимізаторами. Наприклад, у дослідженні [22] пропонують поліпшувати глибинні моделі оптимізаційними алгоритмами (Aquila Optimizer, Capuchin Search, тощо) для відбору ознак, що підвищує точність при застосуванні в мережах Internet of Things [33]. В цілому, спостерігається тенденція, що класичні методи ML не зникають з IDS, а часто стають основою або складовою складніших систем, наприклад, ансамблів або класифікаторів, які включають кілька рівнів.

У дослідженнях регулярно відзначається високий рівень точності, досягнутий сучасними ML-алгоритмами на стандартних тестових наборах даних – нерідко понад 95–99% [79]. Зокрема, застосування глибоких нейронних мереж дозволило значно підвищити виявлення атак типу DoS/DDoS [79], сканування портів, ботнет-діяльності тощо. Проте варто критично оцінювати такі результати, зважаючи на виклики, з якими стикаються існуючі методи. По-перше, моделі глибокого навчання потребують великих обсягів різноманітних даних для надійного навчання; за нестачі даних вони схильні до перенавчання на специфіці набору даних і можуть втратити здатність узагальнювати (generalization) на нові зразки трафіку. По-друге, класичні алгоритми мають обмеження в продуктивності на сучасних високошвидкісних мережах – наприклад, через високу обчислювальну складність SVM у разі великих вибірок виникають затримки, неприйнятні для реального часу. По-третє, деякі методи (особливо ансамблеві та глибокі моделі) є «чорними скриньками» з точки зору інтерпретації [55]: фахівцю складно пояснити, чому саме система віднесла певний трафік до зловмисного. Відсутність зрозумілості ускладнює довіру до таких IDS та пошук слабких місць в них [2; 34]. Нарешті, важливим показником залишається швидкодія й масштабованість: на високонавантажених мережах обробка гігабітних потоків даних потребує оптимізованих моделей. Деякі потужні алгоритми можуть виявитися занадто повільними або вимогливими до пам’яті при розгортанні на практиці. У зв’язку з цим ряд робіт досліджує можливості прискорення – наприклад, спрощення моделей, вибіркову обробку ознак,

розподілену обробку або використання апаратного прискорення для глибокого навчання. Отже, хоча сучасні ML/AI-підходи значно розширили можливості систем виявлення вторгнень, залишаються актуальними питання балансування їх точності, продуктивності та надійності в реальних умовах мережевого середовища.

1.1. Огляд досліджень, пов'язаних із моделюванням та симуляцією мережевих атак в експериментальних середовищах.

Дослідження у галузі IDS тісно пов’язані з питанням генерації та моделювання мережевих атак для потреб навчання і тестування алгоритмів. Оскільки реальні кібератаки важко передбачити та відтворити під контролем, науковці часто застосовують симуляції атак у лабораторних або віртуальних середовищах. Це дозволяє отримати мітки «атака/норма» і необхідні дані для оцінки ефективності системи. Зокрема, широко відомі інструменти та утиліти для генерування трафіку атак типу Denial of Service (DoS) та Distributed DoS (DDoS). Прикладами є атаки на рівні веб-сервера: HULK (HTTP Unbearable Load King) – генератор рандомізованих масових HTTP-запитів; Slowloris – атака повільного вичерпання ресурсів веб-сервера через незавершені з’єднання; Slowhttptest – варіант повільної HTTP-атаки; GoldenEye – ще один інструмент HTTP-флуду; TCP Flood – атака масового надсилання запитів встановлення TCP-з’єднання та інші [83]. Перераховані види атак часто використовуються для наповнення наборів даних або випробування IDS, оскільки вони відображають реальні загрози відмови в обслуговуванні.

У науковій літературі описано численні експерименти, де дослідники розгортають тестове середовище (фізична чи віртуальна мережа з одним або кількома серверами, маршрутизаторами тощо) і здійснюють на ньому контрольовані атаки, щоб зібрати мережеві журнали, пакети чи потоки для подальшого аналізу. Прикладом є робота Amal Al-Harbi та ін. (2021), в якій для виявлення DDoS-атак на веб-сервері було згенеровано чотири різновиди трафіку: Slowloris, Slowhttptest, Hulk та GoldenEye [79]. Дослідники поєднали дані реального часу, отримані через утиліту CICFlowMeter (в режимі моніторингу

живого трафіку), із наявним набором CIC-IDS2017, сформувавши об’єднаний набір даних з ~946 тисяч мережевих потоків [79]. На цьому наборі даних була навчена глибока нейронна мережа, що класифікувала трафік як нормальний або як один з 4 типів DDoS-атаки, досягнувши близько 97% точності[79]. Такий підхід продемонстрував можливість доповнення статичних наборів даних синтетичними атаками для поліпшення виявлення конкретних загроз.

У сучасних дослідженнях у сфері кібербезпеки розробляються програмні засоби для автоматизації аналізу DDoS-інцидентів із використанням ймовірнісних моделей. Зокрема, у роботі [42] описано підхід до застосування баєсівських мереж для формалізації доказів та обчислення ймовірнісних оцінок причетності конкретного вузла до атаки.

Іншим показовим прикладом є створення спеціалізованого набору даних USB-IDS-1 (2021) на базі серії експериментів з DoS-атаками проти веб-сервера Apache [83]. Група дослідників з Університету Санніо (Італія) згенерувала реальний трафік, запускаючи на тестовому сервері різні DoS-атаки (в тому числі Hulk, TCP Flood, Slowloris, Slowhttptest) за різних умов захисту: без захисту, з ввімкненими модулями ReqTimeout, Mod_evasive та Mod_security Apache [83]. В результаті було зібрано багатошарові дані: як мережеві потоки (показники CICFlowMeter), так і вимірювання стану сервера (кількість одночасних з’єднань, використання CPU/RAM, швидкість обслуговування запитів тощо) [83]. Наявність такої експериментальної платформи дозволила авторам проаналізувати реальний вплив атак на продуктивність сервера та ефективність стандартних модулів захисту. Зокрема, з’ясувалося, що деякі атаки з набору даних CIC-IDS2017 хоча і споживають значну пропускну спроможність мережі, проте майже не впливають на роботу добре налаштованого сервера [83]. Зокрема, атаки DoS із CIC-IDS2017 (наприклад, Hulk) виявилися «небольовими» для сучасного веб-сервера: навіть при великому обсязі трафіку сервер продовжував обслуговувати запити без помітного погіршення якості сервісу [83]. Цей результат вказує на обмеженість моделювання атак лише на мережевому рівні – без врахування поведінки цільової системи. USB-IDS-1 запропонував новий підхід: багаторівневе збирання даних про атаку, яке охоплює як мережеві, так і

прикладні аспекти, а також стан активних засобів захисту. Така багаторівнева симуляція покликана підвищити достовірність оцінки IDS. За рахунок цього USB-IDS-1 позиціонується як більш реалістичний бенчмарк для DoS-атак, оскільки всі згенеровані атаки були підтверджено ефективними – тобто реально призводили до деградації сервісу на жертві [83].

Отже, моделювання атак у контрольованих умовах є необхідним елементом досліджень IDS. Воно дозволяє отримати репрезентативні дані для навчання/тестування та перевірити стійкість алгоритмів до конкретних технік нападу. Водночас потрібно враховувати, що штучно згенерований трафік може відрізнятися від трафіку реальних зловмисників [109]. Існує ризик, що модель навчиться виявляти саме характерні риси симуляції (наприклад, періодичність генерації пакетів, обмежену варіативність IP-адрес тощо), і тому погано спрацює на справжніх атаках у мережі. Крім того, експериментальні атаки зазвичай проводяться у відносно простих топологіях (один сервер і кілька нападників), тоді як у реальному Інтернеті атаки DDoS надходять з тисяч розподілених ботів. Ці чинники ускладнюють адаптацію IDS до реального трафіку: модель, успішна в лабораторії, може дати збій у продуктивному середовищі через непередбачені особливості мережі. Тому сучасні дослідники приділяють увагу тому, щоб, по-перше, максимально урізноманітнити симульовані сценарії (кілька різних атак, різні інтенсивності, комбінації атак), а по-друге, комбінувати дані з реальних мереж зі згенерованими. Це має знизити ризик надмірної прив’язаності IDS до специфіки однієї симуляції. У підсумку, симуляція атак є потужним інструментом для розвитку та валідації інтелектуальних IDS, але результативність, досягнута на штучних наборах, повинна перевірятися додатково на більш реалістичних або зовнішніх даних, щоб оцінити здатність моделі узагальнювати знання за межі навчального середовища [17]. Як буде показано далі, такі перевірки виявляють значні проблеми з перенесенням навчених моделей на нові умови, що залишається відкритим викликом у галузі.

Для об’єктивного порівняння методів IDS дослідницька спільнота покладається на публічні набори даних , що містять мережевий трафік з мітками «атака/норма». За останні 20 років було створено кілька поколінь таких наборів

даних. Розглянемо найбільш відомі з них, акцентуючи увагу на їх перевагах та недоліках у контексті застосування ML-алгоритмів.

KDD Cup 1999 та похідні (NSL-KDD). Першим масштабним бенчмарком для IDS став набір KDD Cup 99, створений на основі записів DARPA'98/'99. Він містив близько 4,9 млн мережевих з'єднань, кожне з 41 ознакою, та був розділений на тренувальну і тестову вибірки з різними атаками. Без перебільшення, KDD'99 домінував у дослідженнях IDS протягом 2000-х років, але згодом були виявлені серйозні проблеми його репрезентативності. По-перше, надлишковість і дублікати: майже 78% записів у тренувальній вибірці та 75% у тестовій були дублікатами, що спотворювало результати навчання [59]. Алгоритми легко досягали високої точності, просто запам'ятавши повторювані шаблони, замість того щоб реально навчитися розпізнавати ознаки атак. По-друге, застарілість атак: набір містив атаки лише до 1999 року і не охоплював нових типів шкідливостей (наприклад, ботнети, сучасні DoS-атаки, експлойти веб-застосунків). Вже у 2010-х роках дослідники відзначали, що KDD'99 більше не відповідає сучасному ландшафту загроз [35]. Проте тривалий час не існувало альтернативних спільнотою прийнятих наборів, тому KDD'99 продовжували використовувати для базового порівняння моделей [35]. Покращена версія NSL-KDD (2009 р.) вилучила дублікати та зберегла помірний за розміром піднабір KDD'99, що полегшує тренування алгоритмів [43]. NSL-KDD містить 125973 записів на навчання і 22544 на тест, з 41 ознакою та 5 класами (норма плюс 4 категорії атак) [85]. Попри усунення деяких недоліків KDD, NSL-KDD успадкував його обмеження щодо застарілості. До того ж розподіл класів у NSL-KDD залишається далеким від реального (атаки представлені частіше, ніж у типовому трафіку, відсутні повністю нові типи загроз). Загалом, KDD'99/NSL-KDD є корисними для початкових експериментів, перевірки базових підходів і порівняння з класичними результатами, однак їх результати не можна безпосередньо екстраполювати на сучасні мережі. Високі оцінки точності на цих наборів даних, які часто наводяться у літературі, не гарантують аналогічної ефективності моделі в умовах реального трафіку 2020-х років [85].

UNSW-NB15 (2015). Цей набір даних був запропонований Університетом Нового Південного Уельсу (UNSW, Австралія) як сучасніший бенчмарк, що усуває недоліки KDD'99. UNSW-NB15 зібрано у 2015 р. в тестовій мережі з використанням генератора реального трафіку IXIA, що відтворював нормальну діяльність, та спеціальних скриптів для здійснення 9 типів атак (DoS, порушення цілісності, аналіз, бекдори, експлойти, fuzzers, шпигунське ПЗ, reconnaissance, shellcode, черви) [98]. Набір нараховує ~100 тис. записів з 49 ознаками (у т.ч. новими характеристиками, що відсутні в KDD). Перевагою UNSW-NB15 є наявність більш сучасних атак та поєднання реального фонового трафіку з синтетичними загрозами. Проте і цей набір має низку обмежень. Так, в дослідженнях відзначали, що розподіл класів в UNSW-NB15 лишається незбалансованим і не зовсім відбиває реальні мережеві умови [72]. Деякі типи атак присутні у великій кількості (наприклад, DoS та fuzzing), тоді як інші (бекдор, черв) мають дуже мало зразків, що ускладнює навчання моделей – класи меншин можуть ігноруватися алгоритмами . Крім того, у UNSW-NB15 були виявлені перетинання між тренувальною і тестовою вибірками (схожі потоки зустрічаються в обох), що могло завищувати оцінки точності деяких моделей. В цілому, UNSW-NB15 став важливим кроком уперед, але його складність і неоднорідність (поєднання кількох різних атак) робить його нелегким бенчмарком – моделі часто показують нижчу точність на UNSW-NB15, ніж на CIC-IDS2017 чи NSL-KDD, через складність задачі [34]. Деякі роботи навіть пропонують окремо обробляти підмножини цього набору або застосовувати спеціальні методи балансування і вибору ознак для покращення результатів. Так чи інакше, UNSW-NB15 – корисний для перевірки здатності моделі справлятися з багатокласовою класифікацією в умовах близьких до реальних.

CIC-IDS2017 (2017) та **CSE-CIC-IDS2018** (2018). Ці набори, створені Канадським інститутом кібербезпеки (CIC, Університет Нью-Брансвіка) спільно з іншими організаціями, зараз є одними з найбільш популярних у дослідженнях IDS. CIC-IDS2017 охоплює тиждень симульованого мережевого трафіку (з 3 по 7 липня 2017 р.), де кожен день присвячено певному сценарію атак. Нормальний трафік генерувався реальними волонтерами відповідно до встановлених профілів

поведінки (типовий веб-серфінг, електронна пошта, чат, потокове відео тощо), а атаки включали: портове сканування, DoS та DDoS (з використанням інструментів Hulk, Slowloris, GoldenEye, LOIC та інших), атаки ботнет (зразок Ramnit), спроби злому по протоколу SSH, атаки типу Brute Force на пароль, атаки веб-застосунків (SQL-ін'єкції, XSS) та infiltration (впровадження бекдора в мережу) [79; 86]. Усі атаки підтверджено відбувалися у мережі тестової лабораторії, їх трафік був записаний і розмічений. В результаті CIC-IDS2017 забезпечив дуже різноманітний набір сучасних атак і нормальної активності, що наближається до реального сценарію корпоративної мережі. Цей набір даних містить повні записи потоків (файли PCAP, а також готові вибірки ознак, виділених утилітою CICFlowMeter – у середньому 80 характеристик на потік). Перевагою CIC-IDS2017 є те, що він відображає актуальні на 2017 р. загрози і включає різні типи атак, дозволяючи тестувати моделі як для бінарної класифікації (норма/атака), так і на багатокласовій класифікації конкретних видів нападів. У низці публікацій підкреслюється цінність CIC-IDS2017 як універсального репрезентативного набору для оцінки IDS [34]. Проте водночас було виявлено і його слабкі місця. По-перше, незбалансованість класів: деякі типи атак присутні у відносно невеликій кількості (наприклад, Heartbleed – лише 11 сесій), тоді як інші (DoS Hulk – понад 220 тисяч) домінують. Це створює типову проблему дисбалансу, за якої модель може навчитися ігнорувати рідкісні класи, фокусуючись на більшості [34]. По-друге, як згадувалося, не всі атаки, зафіксовані у CIC-IDS2017, реально ефективні проти сучасних систем – деякі DoS-атаки впливали лише на мережевий канал, але не призводили до відмови сервісу [83]. Крім того, нормальний трафік походив від обмеженої кількості користувачів і комп'ютерів, тому може не охоплювати усього різноманіття реального фону, що відкриває можливість моделей переобладнатися саме на ці зразки «норми». CSE-CIC-IDS2018 є розширенням попереднього набору: він містить дані за 10 днів, зібрані в рамках спільного проєкту з канадською спецслужбою CSE. В цей набір додано нові сценарії атак (включно з різними варіантами DoS/DDoS, операції ботнетів, спам-розсилки, фішинг тощо), а також збільшено обсяг нормального трафіку. Таким чином, CSE-CIC-IDS2018 надає ще

більший за обсягом і різноманітніший матеріал для тренування та тестування IDS. Він вважається одним з найсучасніших відкритих бенчмарків і широко використовується у роботах 2020–2023 рр., особливо при оцінці моделей глибокого навчання [17]. Слабкі сторони в нього загалом подібні до CIC-IDS2017: нерівномірне представлення класів, лабораторний характер даних, відсутність шифрованого трафіку. Проте за рахунок більшого обсягу (мільйони з'єднань) цей набір даних дозволяє будувати глибокі моделі з більшою кількістю параметрів і краще оцінювати їх стійкість.

BoT-IoT (2018). З розгортанням мереж Інтернету речей (IoT) постали нові виклики для IDS, оскільки трафік IoT-пристроїв відрізняється від традиційного і часто є ціллю ботнет-атак. Набір даних BoT-IoT був створений у Австралійському центрі кібербезпеки на базі спеціально змодельованої IoT-мережі, що складалася з різних пристроїв (датчики, камери, розумні побутові прилади) [7]. У тестовій мережі були ініційовані типові для IoT атаки: масовані DDoS (UDP-флуд, TCP-флуд), сканування портів та сервісів, крадіжка даних (Data exfiltration) і keylogging. Загалом Bot-IoT містить мільйони записів, згенерованих утилітами Metasploit та hping. Основна перевага цього набору – акцент на IoT-трафіку: він включає нетиповий для ПК фоновий трафік (MQTT, Telnet, DNS запити від IoT тощо) і специфічні атаки на уразливі IoT-протоколи. Завдяки цьому BoT-IoT дозволяє перевірити, чи здатна IDS виявляти атаки у середовищі розумного дому або промислового IoT. Однак недоліком є вкрай сильний дисбаланс даних: частка нормальних з'єднань мізерно мала порівняно з атакувальними (співвідношення може сягати 0,1%), а серед атак переважають DDoS-флуди [29]. Наприклад, один сценарій DDoS-UDP створив понад 300 мільйонів записів, тоді як інші типи атак – лише тисячі. Це висуває великі вимоги до алгоритмів: моделі мусять коректно працювати при перевантаженні атакувальним трафіком і при цьому не «забувати» про рідкісні класи. Деякі дослідники для спрощення використовують скорочені версії BoT-IoT (підмножини або агреговані по потоках дані, як NF-BoT-IoTv2) [34]. Ці урізані варіанти все одно залишаються складним випробуванням: відомі випадки, коли моделі досягали 99–100% точності на BoT-IoT [34], але автори наголошували на

необхідності перевіряти їх на інших наборах, аби уникнути перенавчання під одну доменну область [34]. Отже, BoT-IoT є важливим для досліджень безпеки IoT, але вимагає спеціальних методів для обробки незбалансованих даних (переважність зразків певного класу, генерація даних меншини, кастомні метрики оцінки тощо).

USB-IDS-1 (2021). Вищезгаданий набір з Університету Санніо вирізняється тим, що націлений на багаторівневий аналіз атак DoS. Він включає лише трафік відмови в обслуговуванні, але з додатковими вимірами – показниками роботи сервера та застосованих механізмів захисту [83]. Перевага USB-IDS-1 – у можливості оцінити, чи справді та чи інша атака була ефективною (наприклад, знизила пропускну здатність сервера, збільшила час відповіді, призвела до збоїв). Це важливо з прикладної точки зору: IDS варто оцінювати не лише за фактом наявності трафіку, схожого на атаку, а й з урахуванням її реальної небезпеки для системи. USB-IDS-1 показав, що деякі «атаки» з CIC-IDS2017 можуть не потребувати негайного спрацьовування IDS, бо не шкодять цільовому сервісу [83]. Водночас атаки, які обминають типові захисні модулі (як Slowloris із увімкненим mod_reqtimeout), є більш критичними і їх слід виявляти в пріоритетному порядку [83]. Недоліком USB-IDS-1 можна вважати вузьку спрямованість – він не містить інших типів атак, окрім DoS проти веб-сервера, тож не підходить для загального тренування багатокласових IDS. Але він є хорошим доповненням до інших наборів, дозволяючи перевіряти адаптивність моделей до включення нових ознак (показників стану системи) та до мультимодального аналізу.

Публічні набори даних відіграють ключову роль у розвитку IDS, проте кожен з них має свої обмеження. Результати досліджень завжди варто інтерпретувати з урахуванням того, на якому наборі даних проводилась оцінка. Висока точність на одному з них (наприклад, 99% на CIC-IDS2017 чи BoT-IoT) не гарантує аналогічної успішності на інших даних або в реальній мережі [17]. Як зауважують Laurens D’hooge та ін. (2023), покращення метрик на популярних академічних датасетах часто переоцінюється – моделі рідко набувають дійсно узагальненого розуміння атак, натомість пристосовуючись до специфіки

конкретного набору [17; 25; 112]. При спробі застосувати такі моделі до трафіку з іншого джерела їх точність суттєво падає [25]; стабільність роботи на нових зразках є проблемною для всіх популярних методів і всіх класів атак [17]. Тому вважається необхідним перевіряти міждатасетну переносимість [58] та розробляти методики domain adaptation [100], які б дозволяли IDS ефективно працювати навіть на тих видах трафіку, які не були представлені в навчальних даних.

Міжнародні датасети та бенчмарки вирішують задачу відтворюваності, однак для практичного застосування IDS не менш критичною є довіра до даних: цілісність, автентичність та неспотвореність потоків/подій, на яких навчаються й тестуються моделі. Так у роботі [89] систематизувано підходи до криптографічної автентифікації й виявлення маніпуляцій у Big Data, що передаються телекомунікаційними каналами, акцентуючи на захисті саме властивостей даних (цілісність/автентичність) у масштабних потоках [89].

1.2. Огляд досліджень багаторівневих класифікаторів та багатоступеневих систем ухвалення рішень

Одним зі шляхів підвищення ефективності IDS, представлених у літературі, є використання багаторівневих (ієрархічних) архітектур класифікації. Ідея полягає в тому, щоб розбити задачу виявлення вторгнень на кілька етапів або рівнів, кожен з яких спеціалізується на своєму підзавданні. Такий підхід може як підвищити точність (за рахунок більш спеціалізовних моделей на кожному рівні), так і зменшити витрати ресурсів (перший рівень відсіює більшість нормального трафіку, далі аналізуються лише підозрілі зразки).

Одним із найбільш розповсюджених варіантів багаторівневої структури – це спочатку бінарний класифікатор, що відділяє весь трафік на дві категорії: «нормальний» і «потенційно атакуючий», а потім багатокласовий класифікатор для детальнішої ідентифікації типу атаки серед підозрілих зразків. Такий підхід використано, наприклад, Qaddoura та ін. (2021) для IDS в IoT-мережах [54]. На першому етапі нейронна мережа визначає сам факт вторгнення, а на другому – інша модель класифікує загрозу як DDoS, ботнет, сканування тощо. Щоб

покращити якість класифікації на другому рівні, автори також застосували техніку oversampling (зокрема SMOTE) для вирівнювання дисбалансу між різними типами атак [54]. Експерименти показали, що дворівневий підхід дав змогу підвищити повноту виявлення рідкісних атак порівняно з одноетапною багатокласовою моделлю, оскільки перший рівень відфільтрував великий обсяг нормального трафіку і модель другого рівня сфокусувалася лише на диференціації між атаками. Аналогічні ідеї зустрічаються і в інших роботах – наприклад, запропоновано спочатку відокремлювати масові DoS-атаки від решти трафіку, а вже потім класифікувати малі атаки окремо, оскільки методи і ознаки, що добре виявляють DoS (потокові статистики, швидкість пакетів), можуть «приглушувати» менш помітні атаки типу U2R або R2L, якщо тренувати все в одній моделі.

У цьому напрямі українські дослідники С. Толюпа, О. Успенський [111] представили підхід до побудови систем захисту на основі багаторівневої ієрархічної моделі, у якій складну систему доцільно розкладати на функціонально відносно незалежні рівні з контролем цілісності при переходах між ними. Така постановка є методично близькою до ідеї розділення детекції/ідентифікації на рівні – коли перший рівень виконує грубе відсівання, а наступні забезпечують деталізацію рішення. [111]

Г. Луцьким та ін. [95] було запропоновано спосіб виявлення атак у реальному часі на основі обчислювального інтелекту, використовуючи генетичне програмування для формування профілів атак. Такий підхід добре узгоджується з ідеєю «першого етапу» як механізму швидкого відбору підозрілих фрагментів/патернів на потоці даних [95].

Також у 2000-х роках з’явились перші гібридні IDS, що поєднували сигнатурний та аномалійний підхід у багаторівневій схемі. Наприклад, Depren et al. (2005) запропонували систему, де на верхньому рівні працював модуль виявлення аномалій (Self-Organizing Map) для відсіювання явних відхилень, а на нижньому – сигнатурний модуль (Decision Tree) для класифікації відомих атак серед виявлених аномалій [16]. Такий підхід зменшував кількість помилкових тривог: все, що не відповідало жодній відомій сигнатурі і водночас не виходило

за межі норми, вважалося доброзичливим трафіком. Інший напрям – комбінування керованого та некерованого навчання. Gogoi et al. (2013) описали Multi-Level Hybrid IDS (MLH-IDS), що складається з трьох компонентів:

- кластеризація для грубого поділу трафіку (щоб виділити великі групи явно нормальних зразків [27]),

- детектування викидів (outlier detection) для ізолювання потенційно аномальних точок,

- класичний супервізований класифікатор для фінальної класифікації залишених підозрілих зразків [27].

Архітектура MLH-IDS передбачала кілька рівнів: спершу дані ділились на три категорії – DoS, Probe та «решта» (в яку входили важко виявні атаки U2R/R2L разом із нормою) [27]. Далі для кожної категорії застосовувалися окремі спеціалізовані алгоритми: для DoS – простіші детектори, для Probe – інші, а для «решти» – додаткові методи виділення ознак і класифікації, щоб розрізнити складні атаки від нормальної активності [27]. Така деревоподібна схема дозволила підвищити загальну точність по всіх класах, особливо покращивши виявлення рідкісних класів U2R/R2L, які часто губляться при плоскому підході [27].

Варто відзначити модель інтелектуального виявлення кібератак на основі логічних процедур та покриттів матриць ознак, запропоновану О. Корченком, В. Лахном та ін. [75]. У роботі формалізовано підхід до побудови системи детекції через формування допустимих та максимальних кон’юнкцій характеристичних ознак класів атак, що дозволяє реалізувати структуровану процедуру прийняття рішень у багаторівневій архітектурі IDS [75]. Зазначений підхід орієнтований на логіко-дискретні механізми розпізнавання та підкреслює доцільність ієрархічної організації процесу аналізу мережевого трафіку.

Також у цьому контексті С. Толюпа та ін. [110] запропонували архітектуру, де у системі виявлення кібератак поєднуються сигнатурні та статистичні аналізатори в межах узгодженого процесу детекції та прийняття рішення. Такий підхід важливий тим, що показує: сигнатурний компонент можна інтерпретувати як «перший бар’єр» (низька затримка, високий пріоритет для відомих патернів),

тоді як статистичний аналізатор – як механізм стабілізації та узагальнення для варіативних або нових проявів атак.

С. Гнатюк та інші [82] запропонували систему виявлення аномалій для оператора стільникового зв'язку за концепцією Big Data, де поєднуються швидкі сигнатурні механізми для відомих загроз та «інтелектуальні» методи для відстеження нетипової активності, а масштабування обґрунтовується через розподілене зберігання та паралельні обчислення [82].

Сучасні тенденції включають побудову адаптивних багаторівневих IDS, які можуть змінювати свою структуру або параметри на льоту, підлаштовуючись під мережеві умови. Наприклад, Yang et al. (2022) запропонували Adaptive Multi-Level Classifier Network (AMLCN) – архітектуру, що динамічно об'єднує класифікатори різної глибини та складності з ваговими коефіцієнтами залежно від поточного трафіку [32]. Ідея AMLCN полягає в тому, що при спокійному стані мережі використовуються лише «верхні» рівні (менш витратні і прості моделі), а при виявленні підозрілих тенденцій автоматично залучаються «глибші» рівні аналізу (наприклад, складні нейронні мережі) з більшим ваговим вкладом у рішення [32]. Таким чином, система прагне балансувати між швидкістю та точністю. Інший підхід – багатоступенева фільтрація з різними методами на кожному етапі. Наприклад, на першому ступені можуть застосовуватися обчислювально прості або евристичні методи (поріг за кількістю з'єднань з однієї IP чи прості евристики) для відсіву явних атак типу сканування чи грубого DoS. Далі на другому ступені застосовується класифікатор на основі ML для детектування менш очевидних атак серед решти трафіку. На третьому ступені – можливо, експертна система чи сигнатурний аналіз для точкової ідентифікації специфічних відомих атак. Кожен ступінь працює з дедалі меншою підмножиною даних, тому можна дозволити більш важкі обчислення. Такі концепції описуються як перспективні, особливо для реалізації у вигляді розподілених систем (напр., перший рівень на рівні мережевого обладнання, другий – у вигляді центрального аналізатора тощо).

У цьому напрямі П. Складанним та ін. [108] було запропоновано модульний підхід на основі нейронних мереж для виявлення різних класів мережевих атак:

окремі модулі опрацьовують групи однотипних параметрів і можуть відключатися/перенавчатися незалежно. Це прямо підтримує ідею багаторівневих систем як «композиції спеціалізованих елементів», а не моноліту [108].

Поміж інших дворівневих моделей в [90] була запропонована модель TP-IDS, яка реалізує описану вище схему: SVM + kNN на першому рівні та Decision Tree + Naive Bayes на другому. Впроваджена в розподіленому середовищі Hadoop, ця система досягла 99,93% точності на тестових даних NSL-KDD, помітно випередивши за точністю однорівневі алгоритми (в тому числі глибокі нейронні мережі) [90]. Особливо важливо, що показники FPR і FNR були значно кращими: за рахунок другого рівня валідації FP та FN знизились, а загальна точність зросла [90]. Система виявилась здатною в реальному часі обробляти великий потік даних завдяки паралельності та ефективному відсіюванню нормального трафіку на першому етапі [90].

У роботі Ahamed & Karim (2025) запропоновано Cascaded IDS (CIDS), що поєднує сигнатурний підхід (дерево рішень) та повністю однокласовий метод (OC-SVM) для виявлення нових атак [76]. Ця дворівнева гібридна система продемонструвала покращену здатність виявляти як відомі, так і невідомі загрози. Згідно з результатами, CIDS перевершила окремо взяті сигнатурні та аномалійні детектори: вона успішно виявляє типові мережеві атаки і одночасно класифікує невідомі раніше атаки з високою точністю [76]. Кількість хибних спрацювань також зменшилась порівняно з традиційними IDS завдяки каскадній перевірці та новому модулю класифікації на другому рівні [76]. Авторами підкреслено, що об’єднання переваг сигнатурного та поведінкового аналізу в межах дворівневої моделі дає стійку та масштабовану відповідь сучасним викликам кібербезпеки [76].

У недавньому дослідженні Uddin та ін. (2024) запропоновано адаптивну дворівневу IDS з використанням однокласових класифікаторів (OCC) для першого етапу та поєднанням кластеризації і наглядового навчання на другому [44]. На першому рівні модель навчається виключно на нормальних даних і виявляє будь-які відхилення як потенційні атаки, що знімає проблему

дисбалансу (мало зразків атак) [44]. На другому рівні всі виявлені аномалії класифікуються: відомі типи атак розпізнаються Random Forest-класифікатором, а невідомі групуються кластеризацією для подальшого навчання системи [44]. Така система постійно адаптується до нових загроз, донавчаючи моделі по мірі накопичення невпізнаних випадків. Результати показали, що дворівнева OCC-архітектура ефективно виявляє як відомі, так і нові атаки, мінімізуючи частку пропущених загроз [44]. Цей підхід демонструє перспективність багаторівневого виявлення для динамічних середовищ (наприклад, IoT), де сигнатури швидко застарівають і потрібне постійне навчання системи.

Узагальнюючи, багаторівневі та багатоступеневі архітектури IDS мають такі переваги:

- кожен рівень може бути налаштований під свій набір ознак чи тип атак, що підвищує загальну точність (як показали приклади з окремим обробленням DoS та складних атак) [27].

- попереднє відсіювання нормального трафіку або простих випадків знижує обсяг даних для детального аналізу, що покращує швидкодію і дозволяє застосувати складні методи тільки там, де це потрібно.

- можна легко додавати нові рівні або модулі під нові загрози, не перебудовуючи повністю всю систему. Також, у разі адаптивних схем, система може реагувати на зміни (наприклад, при виявленні несподіваних аномалій додатково включати глибший аналіз).

Водночас такі моделі мають ряд недоліків та складнощів, які перелічені нижче:

- якщо перший рівень припустився помилки (пропустив атаку або помилково заблокував нормальний трафік), подальші рівні вже не виправлять цього – хиба «поширюється» по ланцюгу. Тому критично важливо, щоб початкові фільтри мали високий recall (не пропускали атак).

- у багатоступеневих схемах часто є параметри порогів чи умов переходу між рівнями (наприклад, яке значення метрики аномальності вважається достатнім для передачі далі). Невірний вибір цих порогів може або

перевантажити наступні рівні зайвими даними, або навпаки – відфільтрувати корисну інформацію.

- багаторівнева система є більш складною у розробці та підтримці. Потрібно забезпечити узгоджену роботу всіх модулів, розробити механізм інтеграції їх рішень (пріоритети, логіку «голосування» і т.д.). Також зростає складність тестування, адже треба перевіряти не лише кожен компонент, а й їх взаємодію на системному рівні.

- хоч окремі рівні можуть бути оптимізовані, але сумарно проходження кількох етапів (особливо якщо вони послідовні) додає затримку до часу виявлення. В критичних застосуваннях реального часу надлишкова затримка може виявитися неприйнятною.

У дослідженнях IDS на основі виявлення аномалій пороги часто описуються як «слабка ланка» між теорією моделі та практикою експлуатації, оскільки вони визначають межу між нормою, підозрою та атакою в умовах невизначеності. Монографія А. Корченко [74] присвячена методам ідентифікації аномальних станів для систем виявлення вторгнень, систематизує підходи до опису аномальних станів як об’єктів, що можуть бути виміряні, класифіковані й інтерпретовані (зокрема через нечіткі/лінгвістичні форми подання), і тим самим задає рамку для порогових рішень як частини керованого процесу.

Питання порогів дедалі частіше поєднують із темою адаптації до змін середовища та зсуву розподілів ознак. У цьому напрямі В. Лахном [94] було запропоновано підхід до побудови адаптивної системи розпізнавання кіберзагроз на основі нечіткого кластеризування ознак, у межах якого правила/межі класифікації можуть коригуватися в процесі експлуатації. Важливо, що адаптація у такій рамці описується не як «разове перенавчання», а як механізм керованої еволюції рішень (зокрема через корекцію правил), що близький до практичної задачі підтримки працездатності при зміні трафіку [94].

Попри ці виклики, огляд літератури свідчить, що багаторівневі підходи набувають популярності при проєктуванні «розумних» IDS. Особливо це стосується середовищ зі строгими вимогами – таких як промислові мережі чи IoT, де бажано мінімізувати як помилкові спрацьовування, так і пропущені атаки.

Наприклад, у сфері промислових систем управління (ICS) пропонуються багаторівневі IDS, де перший рівень – білий список дозволених команд, другий – поведінкова модель МН, що стежить за аномаліями у послідовності команд, а третій – модуль, що застосовує сигнатури відомих шаблонів атак на ПЛК. Така комбінація забезпечує глибший аналіз з урахуванням технологічного процесу. Загалом, багаторівневі системи розглядаються як спосіб об'єднати переваги різних методів та компенсувати їх недоліки один одним у межах єдиної архітектури [27].

Переваги, недоліки та методологічні прогалини у дослідженнях IDS.

На основі проведеного огляду можна виокремити ряд переваг сучасних наукових підходів до створення інтелектуальних IDS, а також їх недоліків і відкритих проблем (прогалин), що потребують уваги дослідників.

Завдяки застосуванню ML/DL-методів багато робіт демонструють успішне виявлення більшості поширених атак з точністю 95–99% [79] на тестових наборах. Згорткові та рекурентні нейромережі виявилися здатними розпізнавати складні шаблони атак, недоступні для простих евристик. Ансамблеві методи знижують імовірність помилки окремого класифікатора, підвищуючи надійність. Отримано позитивні результати в виявленні DoS/DDoS, портсканів, атак на веб-застосунки, деяких типів шкідливого ПЗ тощо. Загалом, дослідження підтвердили принципову придатність технологій ШІ для задач кібербезпеки, що є значним кроком уперед порівняно з традиційними сигнатурними системами.

Окрім безпосереднього застосування відомих моделей (таких як Random Forest чи CNN), ряд робіт запропонували нові, оптимізовані під IDS рішення: спеціалізовані архітектури нейромереж для трафіку, алгоритми оптимізації гіперпараметрів (генетичний алгоритм, рій часток, метод колонії мурах) спеціально для налаштування IDS[34], гібридні алгоритми (наприклад, об'єднання двох методів класифікації з різною природою). Ці напрацювання збагачують набір інструментів і наближають нас до створення більш «інтелектуальних» систем, здатних навчатися ефективніше.

Якщо раніше фокус був переважно на класичних мережах і інтернет-трафіку, то сучасні дослідження охоплюють також специфічні сфери: безпека IoT

(як-от датасети Bot-IoT, NF-BoT-IoT) [34], безпека промислових мереж (є окремі датасети та емулятори для SCADA-систем), хмарні середовища та віртуалізація. Це дозволяє адаптувати IDS до нових сценаріїв застосування – з'являються методи для захисту інтелектуальних будинків, автомобільних мереж, контейнерів у хмарі тощо.

У деяких оглядових роботах особливо відзначається аспект атак з обходу IDS (наприклад, шифрування трафіку, поліморфізм та інші методи маскування) [35]. Пропонуються підходи підвищення стійкості моделей – використання більш стійких ознак (наприклад, статистичних характеристик трафіку, що важко підробити), впровадження моделей з інтерпретованими рішеннями (щоб оператор міг зрозуміти, чи не була модель обдурена маніпуляціями). Деякі праці, хоч і небагато, почали розглядати контраварійні методи – наприклад, навчання на adversarial examples (штучних зразках, спотворених спеціально для обману моделі) з метою зробити IDS міцнішою до таких атак. Це свідчить про зрілість галузі: увага звертається не лише на виявлення «чесних» атак, але й на боротьбу з тими, хто намагається обманути саму систему захисту.

Поява публічних наборів даних і метрик (Accuracy, Precision, Recall, F1, ROC-AUC тощо) дозволила об'єктивніше порівнювати методи між собою. Багато оглядів структурують результати різних алгоритмів на спільних наборах – це допомагає визначити, які підходи є найбільш перспективними. Наприклад, систематичний аналіз 393 статей виявив лідируючі алгоритми (CNN, SVM, DT, GA) і підтвердив тренд до зростання публікацій саме в цих напрямах [34]. Таким чином, поступово формується певний консенсус щодо найкращих практик (наприклад, використання поєднання детектора аномалій і класифікатора атак, застосування методів балансування даних перед навчанням моделей тощо).

Разом з тим більшість досліджень, пов'язаних з використанням моделей машинного навчання мають ряд недоліків та методологічних прогалин.

Одним з найсерйозніших викликів є вже згадана здатність моделей генералізувати знання на нові середовища, поза тими даними, на яких вони навчалися. Більшість наукових робіт перевіряють алгоритми тільки на одному-двох датасетах, причому, як правило, навчання і тестування відбувається на

частинах одного датасету (random split). При цьому дуже рідко ставиться питання: чи означають відмінні результати на цьому наборі, що модель так само добре спрацює на іншому? Дослідження D'hooge et al. (2023) дало тривожну відповідь: висока точність на тестовій вибірці CIC-IDS2018 майже не корелює з успіхом на інших наборах (CIC-IDS2017, CIC-DoS2017, CIC-DDoS2019) для тих самих типів атак [17]. Автори виявили, що моделі рідко вчаться «по-справжньому» розпізнавати атаку, натомість підлаштовуються під специфічні особливості саме цього набору даних. При перенесенні, наприклад, детектора DoS з CIC-IDS2018 на CIC-2017, точність різко падала, хоч атака номінально та ж сама [17]. Основний висновок – не можна очікувати, що навіть найкращі на сьогодні моделі, натреновані на академічних датасетах, дадуть аналогічний результат у реальній мережі без додаткового донавчання або адаптації [17]. Це означає, що узагальненість залишається відкритою проблемою. Для її вирішення пропонуються напрямки майбутніх робіт: використання кількох різнорідних датасетів при навчанні (щоб модель бачила більше варіацій атак), методи transfer learning та domain adaptation, періодичне переонавчання IDS на нових даних з реальної мережі (онлайн-навчання) тощо.

Як описано вище, практично всі реальні датасети IDS є незбалансованими: нормальний трафік або певні поширені атаки значно переважають інші класи. Це спричиняє зміщення моделей у бік більшості і погане виявлення рідкісних, але критичних загроз. Наприклад, модель може досягти 99% точності, фактично ігноруючи малочисельний клас (так званий ефект «Accuracy Paradox»). У оглядах це названо однією з головних методологічних проблем [34]. Для боротьби з дисбалансом у дослідженнях застосовують методи балансування вибірки (оверсемплінг меншості – як у Qaddoura et al. 2021 [54], андерсемплінг більшості, синтетичне створення зразків атак), а також спеціальні метрики оцінки (матриці невідповідностей, F1-міра по кожному класу, оцінка G-Mean і т.д.). Однак повністю вирішити цю проблему складно, оскільки реальні атаки за визначенням рідкісні. В академічних датасетах інколи навпаки збільшують частку атак, щоб моделі могли навчитися – але тоді IDS не відчуває «навантаження» нормального трафіку. Отже, проблема дисбалансу напряму пов'язана зі здатністю моделі

працювати в реальному часі: при дуже низькому відношенні «атака/норма» алгоритми можуть генерувати забагато помилкових спрацювань, якщо не були належно навчені. Це поле для подальших досліджень – наприклад, застосування спеціалізованих алгоритмів класифікації, стійких до дисбалансу (One-class SVM, Isolation Forest, cost-sensitive learning).

Більшість поточних рішень IDS працюють в статичному режимі: модель тренується офлайн на певному датасеті, після чого використовується для детекції. Але мережевий трафік і методи атак постійно еволюціонують (концепція concept drift). Модель, розгорнута сьогодні, через півроку може зіткнутися з новими типами легітимного трафіку (наприклад, сплеск відеоконференцій) або з новими видами сканерів/експлойтів, про які вона не знає. В такому випадку точність неминуче впаде. Наукові роботи поки що рідко моделюють цей аспект.

Також, як вже зазначалося вище, сучасні моделі – особливо глибокі – є складно інтерпретованими. В галузі кібербезпеки це критично: аналітики SOC (центру моніторингу безпеки) повинні розуміти причину спрацювання IDS, щоб прийняти подальші дії. Якщо система видає алерт «аномалія в трафіку», але не пояснює, які саме пакети чи параметри викликали підозру, фахівцю важко оцінити його значущість. Тому нині з’являється тенденція до Explainable AI (XAI) в IDS: використання методів, що надають пояснення (наприклад, застосування LIME/SHAP для нейромереж, візуалізація найважливіших ознак, побудова сурогатних інтерпретованих моделей поруч з основною тощо). Поки що таких робіт небагато, і більшість високоточних моделей залишається «чорними ящиками». Це недолік, що гальмує впровадження ML-IDS в практику, адже організації не готові довіряти системі, рішення якої незрозумілі.

Деякі підходи, які чудово працюють на вибірках розміром десятки тисяч з’єднань, можуть не масштабуватися на реальні потоки даних у гігабітній мережі. Наприклад, глибока нейромережа може вимагати GPU і значного часу на інференс для кожного пакету, що нереально при обробці мільйонів пакетів за секунду. Питання оптимізації та розгортання ML-моделей в режимі реального часу часто опускається в наукових статтях – це своєрідний розрив між теорією і

практикою. У якості прогалин можна вказати відсутність тестування на пропускну здатність (throughput) IDS, на затримку виявлення (time-to-detect) та на стійкість до високонавантажених сценаріїв. Також мало уваги приділено розподіленим архітектурам: як запустити ту саму модель одночасно на декількох вузлах мережі, щоб покрити великий трафік. Ці питання залишаються для майбутніх досліджень, можливо, на стику з інженерією масштабованих систем.

Підсумовуючи, сучасні дослідження дали спільноті потужний арсенал методів для виявлення вторгнень і підтвердили перспективність інтеграції штучного інтелекту в системи захисту. Водночас, існує розуміння, що деякі фундаментальні проблеми ще не вирішено. Обмеження наявних датасетів (їхня обмежена кількість, синтетичність, незбалансованість) та відсутність перевірки моделей «в полі» породили ситуацію, коли чимало запропонованих рішень чудово працюють в лабораторії, але їх практична цінність сумнівна [17]. Наступні кроки мають бути спрямовані на подолання цих прогалин: створення більш реалістичних відкритих датасетів (наприклад, з включенням реального трафіку і атак, що дійсно впливають на сервіси) [98], розробка методів тренування моделей на обмежених та нерівномірних даних [34], дослідження методик оновлення моделей без потреби повного перевчання, а також забезпечення прозорості і довіри до рішень IDS (Explainable IDS).

1.3. Інструментальні системи виявлення вторгнень: аналіз сучасних рішень

Системи виявлення вторгнень у практичних мережевих середовищах формувалися як інженерні рішення, орієнтовані на стабільність, передбачуваність і контрольованість поведінки. Незважаючи на активний розвиток алгоритмів машинного навчання, у промислових інфраструктурах домінують системи, що реалізують сигнатурну або гібридну логіку детекції. Такі інструменти виконують роль первинного рівня захисту, забезпечуючи фільтрацію трафіку, формування подій безпеки та інтеграцію з системами моніторингу.

У цьому підрозділі здійснюється послідовний і порівняльний аналіз найбільш поширених інструментальних рішень у сфері IDS/IPS, з акцентом на їх

архітектурні принципи, функціональні можливості, експлуатаційні переваги та системні обмеження. Розгляд здійснюється без апеляції до конкретних алгоритмів машинного навчання, з метою об’єктивного визначення стану галузі та виявлення її концептуальних прогалин.

«Snort» є однією з найвідоміших відкритих мережевих систем виявлення вторгнень, яка історично сформувала основу сигнатурного підходу до детекції мережевих атак. Система функціонує як мережевий сенсор, що перехоплює пакети трафіку, декодує їх відповідно до протоколів канального, мережевого та транспортного рівнів, після чого здійснює перевірку на відповідність заздалегідь визначеним правилам. Архітектура «Snort» побудована за модульним принципом і включає декодер пакетів, препроцесори, рушій детекції та механізм формування логів [92].

Препроцесори відіграють важливу роль у нормалізації трафіку, оскільки здійснюють реконструкцію TCP-потоків, дефрагментацію IP-пакетів та усунення технік обходу, пов’язаних із маніпуляцією сегментацією. Це дозволяє зменшити ризик хибнонегативних спрацювань, спричинених спеціальною модифікацією пакетної структури зловмисником. Таким чином, «Snort» виконує не лише просте зіставлення сигнатур, а й попередню обробку трафіку з урахуванням можливих технік ухилення.

Рушій детекції використовує набір правил, кожне з яких формалізує шаблон відомої атаки. Правила описують умови відповідності за IP-адресами, портами, напрямком з’єднання, а також специфічними байтовими послідовностями в корисному навантаженні. Така структура забезпечує високу інтерпретованість: кожне спрацювання має чітке пояснення у вигляді конкретного правила, що активувалося.

Сильним боком «Snort» є прозорість прийняття рішення та контрольованість логіки роботи. Адміністратор може детально налаштовувати правила, визначати рівень чутливості, відключати або модифікувати окремі сигнатури. Це забезпечує гнучкість у конкретному середовищі розгортання та дозволяє адаптувати систему під специфіку організації.

Однак сигнатурна природа «Snort» обмежує його здатність до виявлення нових або модифікованих атак. Система не має механізму узагальнення або статистичного аналізу поведінки, тому будь-яка нова загроза, що не відповідає наявним правилам, залишиться непоміченою [6]. Ефективність системи безпосередньо залежить від регулярності оновлення бази сигнатур.

Крім того, зростання частки шифрованого трафіку знижує інформативність аналізу [31] на рівні корисного навантаження, якщо не використовується SSL/TLS-інспекція. У багатьох середовищах розшифрування трафіку є обмеженим через політичні або юридичні причини, що додатково зменшує потенціал сигнатурної детекції.

Таким чином, «Snort» є ефективним інструментом для контролю відомих загроз у стабільному середовищі, але не забезпечує адаптивності або формалізованої оцінки невизначеності в умовах змінної мережевої поведінки.

«Suricata» розроблена як високопродуктивна мережева IDS/IPS із підтримкою багатопотокової обробки трафіку. На відміну від класичної архітектури «Snort», «Suricata» реалізує multi-threaded engine, що дозволяє масштабувати систему відповідно до кількості ядер процесора та забезпечувати ефективну роботу у високошвидкісних мережах [77].

Архітектура «Suricata» включає модуль захоплення пакетів, рівень глибокого аналізу протоколів (Deep Packet Inspection), рушій сигнатурної детекції та систему формування структурованих журналів подій [60]. Підтримка DPI дозволяє аналізувати прикладні протоколи, зокрема HTTP, DNS, TLS, FTP, що розширює контекст детекції та підвищує точність виявлення відомих загроз.

Однією з суттєвих переваг «Suricata» є формат журналів EVE JSON, який забезпечує структуроване подання подій безпеки та спрощує інтеграцію з системами централізованого моніторингу і SIEM-платформами [114]. Це дозволяє використовувати «Suricata» як повноцінний сенсор телеметрії в комплексних архітектурах безпеки.

«Suricata» підтримує як режим IDS, так і режим IPS із можливістю блокування трафіку в реальному часі [51]. Завдяки багатопотоковій обробці

система здатна обробляти значні обсяги трафіку без критичних затримок, що є важливим для сучасних мереж із гігабітною пропускною здатністю.

Попри технічну досконалість, «Suricata» зберігає сигнатурну основу детекції. Логіка прийняття рішення базується на правилах, а не на ймовірнісних або адаптивних моделях. Вона не реалізує формалізованої оцінки ризику або автоматичного коригування порогів залежно від зміни статистичного профілю трафіку.

Як і у випадку «Snort», ефективність «Suricata» значною мірою залежить від актуальності правил та регулярного адміністрування. Крім того, обробка шифрованого трафіку залишається обмеженою без додаткових механізмів розшифрування [51].

Таким чином, «Suricata» є інженерно оптимізованою та масштабованою сигнатурною IDS/IPS, що забезпечує високу продуктивність і зручність інтеграції, але концептуально не виходить за межі парадигми на основі правил.

«Zeek» («Bro») є системою мережевого моніторингу, що реалізує подієву модель аналізу трафіку. На відміну від класичних IDS, «Zeek» не обмежується генерацією сповіщень, а формує детальні журнали мережевих взаємодій, відображаючи семантичний контекст прикладних протоколів [52].

Архітектура «Zeek» включає механізм аналізу пакетів, реконструкцію сесій та генерацію подій, які інтерпретуються через скриптову мову політик. Це дозволяє формалізувати складні сценарії детекції та адаптувати систему до специфіки організації.

Сильним боком «Zeek» є глибина контекстного аналізу. Система формує окремі журнали для HTTP, DNS, TLS, SMTP та інших протоколів, що забезпечує високий рівень деталізації мережевої поведінки. Такий підхід робить «Zeek» цінним джерелом даних для подальшої аналітики або машинного аналізу.

«Zeek» добре інтегрується з SIEM-системами та платформами кореляції подій, що дозволяє використовувати його як компонент більш складної архітектури безпеки. Він забезпечує тривалий зберігання історії мережевої активності та підтримує розширення функціоналу через додаткові скрипти.

Водночас «Zeek» не є класичною IDS із централізованим механізмом прийняття рішення про вторгнення. Його ефективність залежить від якості налаштованих політик і від подальшої кореляції журналів. Відсутність внутрішньої ймовірнісної моделі обмежує його здатність до автоматичної адаптації [52].

Таким чином, «Zeek» є потужним інструментом мережевого моніторингу з високим рівнем контекстуалізації, проте потребує інтеграції з іншими компонентами для реалізації повноцінної системи детекції.

«OSSEC» (та «Wazuh») належить до класу Host-Based Intrusion Detection Systems і функціонує на рівні кінцевих вузлів [61]. Система аналізує журнали операційної системи, контроль цілісності файлів, зміну конфігурацій та інші локальні події, що дозволяє виявляти внутрішні компрометації .

«Wazuh» є розвитком «OSSEC» та розширює його функціональність, додаючи централізоване управління агентами, інтеграцію з Elastic Stack та розширені механізми кореляції [12]. Агентна модель дозволяє розгортати систему на великій кількості хостів і централізовано збирати події.

Перевагою HIDS-підходу є можливість виявлення атак, які не залишають явних слідів у мережевому трафіку, зокрема локальні privilege escalation або зміну критичних файлів [45].

Водночас управління великою кількістю агентів створює додаткове навантаження на інфраструктуру, а кореляція великої кількості подій може призводити до інформаційного шуму. Крім того, відсутність мережевого контексту обмежує здатність системи до повного аналізу складних розподілених атак.

Таким чином, «OSSEC»/«Wazuh» є важливим доповненням до мережевих IDS, але не можуть розглядатися як автономне рішення для комплексного захисту.

«Security Onion» є інтеграційною платформою, що поєднує мережеві та хостові сенсори в єдине середовище збору та аналізу подій [3]. Вона включає «Suricata», «Zeek» та інші компоненти, формуючи повноцінний стек для організації SOC [63].

Архітектура платформи орієнтована на централізоване збирання, зберігання та візуалізацію подій безпеки. Перевагою є можливість кореляції різнорідної телеметрії та комплексний огляд стану мережі.

Водночас система не реалізує власної моделі детекції, а інтегрує існуючі інструменти [70]. Її розгортання потребує значних ресурсів і кваліфікованого персоналу.

«Security Onion» є інструментом інтеграції та операційної організації моніторингу, але не усуває фундаментальні обмеження сигнатурних або сценарних механізмів детекції.

«Cisco Secure Firewall» (з рушієм «Snort»), раніше відомий як «Cisco Firepower», є комерційною платформою класу Next-Generation Firewall (NGFW), яка інтегрує функції міжмережевого екрану, системи запобігання вторгненням (IPS), контролю додатків та глибокої інспекції трафіку [30]. Центральним компонентом механізму виявлення вторгнень у цій платформі є рушій «Snort», адаптований і оптимізований для роботи в межах корпоративної екосистеми Cisco.

Архітектурно рішення поєднує сигнатурну детекцію, керовані політики доступу та централізоване адміністрування через систему управління (Cisco FMC – Firepower Management Center). Рушій Snort виконує аналіз трафіку на відповідність сигнатурам, однак у рамках NGFW він функціонує як частина більш складного політичного контуру, де рішення приймається не лише на основі окремого правила, а з урахуванням профілю політики безпеки.

Однією з ключових особливостей «Cisco Firepower» є інтеграція IPS-функціоналу безпосередньо у периметровий міжмережевий екран. Це дозволяє здійснювати inline-блокування трафіку на основі сигнатурних спрацювань, що забезпечує не лише детекцію, а й активне запобігання вторгненням [99]. Крім того, система підтримує SSL/TLS-інспекцію, що дозволяє здійснювати глибокий аналіз зашифрованих з’єднань за умови відповідної конфігурації.

Перевагою платформи є висока продуктивність та оптимізація під апаратні рішення Cisco. Система здатна обробляти значні обсяги трафіку без істотної деградації пропускної здатності, що критично для великих корпоративних

мереж. Крім того, централізоване управління політиками спрощує адміністрування та аудит.

Водночас концептуально «Cisco Firepower» залишається сигнатурно орієнтованою системою. Незважаючи на інтеграцію додаткових аналітичних механізмів (наприклад, Talos Intelligence Feed), основою детекції є оновлювані сигнатури та політики. Система не реалізує відкритої формалізованої моделі оцінки невизначеності або адаптивного порогового прийняття рішень.

Ще одним обмеженням є закритість внутрішньої логіки оптимізації. На відміну від відкритих IDS, адміністратор не має повного доступу до внутрішніх алгоритмів обробки, що ускладнює глибоку кастомізацію або інтеграцію з експериментальними аналітичними модулями.

Таким чином, «Cisco Firepower» є інженерно зрілим та високопродуктивним периметровим рішенням із інтегрованою сигнатурною IPS-функціональністю, однак концептуально зберігає класичну парадигму детекції на основі правил.

«Palo Alto Networks Threat Prevention» є компонентом new generation firewall Palo Alto, що реалізує функції виявлення та запобігання вторгненням, контролю додатків і аналізу шкідливого трафіку [28]. Система функціонує у складі апаратно-програмного комплексу, орієнтованого на глибоку інспекцію трафіку та централізоване управління політиками безпеки.

Архітектурно рішення базується на концепції App-ID, User-ID та Content-ID, що дозволяє здійснювати аналіз не лише на рівні портів або IP-адрес, а й на рівні конкретних додатків та користувачів. Механізм Threat Prevention використовує сигнатури загроз, включаючи мережеві експлойти, malware та C2-комунікації, для детекції відомих атак.

Перевагою платформи є глибока інтеграція політик доступу, аналізу додатків і IPS-функціоналу. Це дозволяє формувати комплексні правила, що враховують контекст взаємодії, тип додатка та профіль користувача. Крім того, система підтримує SSL-дешифрування, що забезпечує доступ до прикладного рівня у зашифрованих каналах.

Суттєвим плюсом є регулярне оновлення сигнатур через хмарну інфраструктуру постачальника. Це дозволяє оперативно реагувати на нові загрози в межах сигнатурної парадигми [28].

Водночас, як і більшість NGFW-рішень, «Palo Alto Networks Threat Prevention» залишається орієнтованою на сигнатурну логіку детекції. Хоча компанія декларує використання аналітичних механізмів і хмарної розвідки загроз, відкритої формалізованої ймовірнісної моделі прийняття рішень не надається.

Крім того, закритість архітектури обмежує можливість глибокої інтеграції із зовнішніми експериментальними аналітичними модулями. ML-компоненти, якщо вони присутні, функціонують як частина внутрішньої екосистеми, а не як відкритий алгоритмічний модуль.

Таким чином, «Palo Alto Networks Threat Prevention» є комплексною системою з високим рівнем інтеграції та керованості, але концептуально продовжує сигнатурно-орієнтований підхід до виявлення загроз.

«Darktrace» є представником класу Network Detection and Response (NDR) та позиціонується як система поведінкової детекції, що використовує методи штучного інтелекту та машинного навчання для моделювання «нормальної» активності мережі. На відміну від сигнатурних IDS, «Darktrace» декларує несигнатурний підхід, орієнтований на виявлення відхилень від базового профілю поведінки [48].

Архітектура системи передбачає пасивний моніторинг мережевого трафіку та формування моделей поведінки користувачів, пристроїв і сервісів. На основі зібраної телеметрії формується динамічний профіль «норми», з яким порівнюється поточна активність [48].

Перевагою такого підходу є потенційна здатність виявляти раніше невідомі або модифіковані атаки, які не мають відповідних сигнатур. Система може фіксувати нетипові шаблони взаємодії, незвичні обсяги передачі даних або аномальну поведінку пристроїв.

Водночас поведінкова модель є чутливою до зміни статистичного профілю середовища. Зміни у бізнес-процесах, міграція сервісів або сезонні коливання активності можуть призводити до зростання кількості помилкових спрацювань.

Ще одним обмеженням є непрозорість внутрішніх алгоритмів. Як комерційна платформа, «Darktrace» не надає детального опису математичних моделей, що ускладнює формальну верифікацію її рішень. Таким чином, «Darktrace» представляє інший концептуальний підхід – поведінкову детекцію з елементами ML, але водночас стикається з проблемами адаптації до змінного середовища та обмеженої інтерпретованості.

«Vectra» є ще одним представником класу NDR, що спеціалізується на виявленні атак на основі аналізу мережевих метаданих та поведінкових патернів [15]. На відміну від систем, що фокусуються на корисному навантаженні пакетів, «Vectra» акцентує увагу на аналізі взаємодій між хостами, сервісами та користувачами.

Система формує профілі поведінки та здійснює класифікацію подій на основі внутрішніх моделей штучного інтелекту. Підхід орієнтований на виявлення lateral movement, C2-комунікацій та інших складних сценаріїв атак.

Перевагою є можливість роботи без повного доступу до корисного навантаження, що є актуальним в умовах широкого використання шифрування. Аналіз метаданих дозволяє зменшити залежність від SSL-дешифрування.

Водночас, як і у випадку інших NDR-рішень, внутрішні моделі є закритими, що обмежує можливість зовнішньої експертної оцінки алгоритмічної коректності.

Крім того, поведінкові системи можуть демонструвати нестабільність у середовищах із високою динамікою, де «норма» швидко змінюється.

Таким чином, «Vectra» представляє еволюцію IDS у напрямі поведінкової та аналітичної детекції, однак стикається з викликами адаптивності та прозорості.

Висновки до першого розділу

Проведений аналітичний огляд висвітлив сучасний стан та тенденції в галузі створення інтелектуальних систем аналізу мережевих атак. Методи машинного навчання та штучного інтелекту вже міцно закріпилися як основа багатьох пропозицій щодо IDS, демонструючи високу точність виявлення різноманітних атак завдяки здатності автоматично навчатися ознакам вторгнень [34]. Класичні алгоритми (наївний Баєс, дерева рішень, SVM тощо) заклали підґрунтя, а сучасні глибокі нейронні мережі та ансамблі вивели якість детекції на новий рівень. Розглянуто дослідження, в яких шляхом симуляції DoS/DDoS-атак (Slowloris, Hulk, TCP Flood та ін.) в тестових середовищах створюються реалістичні датасети для тренування та оцінки IDS [79; 83]. Особливу увагу приділено аналізу публічних наборів даних: від застарілих KDD’99 до сучасних CIC-IDS2017, CSE-CIC-2018, UNSW-NB15, BoT-IoT, USB-IDS-1. Показано, що кожен з них має як сильні сторони (репрезентативність певних атак, великий обсяг даних, реальність сценаріїв), так і слабкі (дубльовані записи, дисбаланс класів, відрив від актуального трафіку) [35; 98]. Розглянуто перспективні архітектури IDS з багаторівневою класифікацією і багатоступеневим ухваленням рішень, які прагнуть підвищити ефективність системи за рахунок розподілу завдань між кількома спеціалізованими модулями [27; 54].

Аналіз сучасних інструментальних систем виявлення вторгнень показав, що у практичних мережевих середовищах домінують сигнатурні та гібридні архітектури, орієнтовані на стабільність функціонування, масштабованість і інтеграцію з інфраструктурою моніторингу безпеки. Такі системи забезпечують глибоку інспекцію мережевих протоколів, підтримку багатопотокової обробки трафіку та централізоване управління подіями безпеки. Поведінкові рішення класу NDR розширюють можливості детекції за рахунок аналізу відхилень від профілю типової мережевої активності. У цілому, сучасні IDS характеризуються високим рівнем інженерної реалізації та здатністю ефективно функціонувати в реальних експлуатаційних умовах.

Разом із тим встановлено, що механізми прийняття рішень у більшості існуючих систем реалізуються на основі правил, сигнатур або внутрішніх

евристичних налаштувань. Перехід між етапами аналізу, налаштування порогів спрацювання та оброблення невизначених ситуацій зазвичай визначаються конфігураційними параметрами та політиками безпеки. Формалізація цих механізмів у вигляді єдиної математичної моделі прийняття рішень із кількісним урахуванням зміни статистичних характеристик трафіку в існуючих рішеннях, як правило, не декларується.

Загальний висновок полягає в тому, що сучасні наукові досягнення створюють міцне підґрунтя для побудови «розумних» IDS, здатних проактивно виявляти широкий спектр кібератак. Проте, існує низка методологічних проблем, які потребують вирішення, перш ніж такі системи зможуть повною мірою реалізувати свій потенціал у реальних умовах. Серед них – обмежена здатність моделей до узагальнення знань за межі навчальних даних, проблема незбалансованих вибірок і рідкісних атак [88], відсутність механізмів безперервної адаптації до нового трафіку, а також питання пояснюваності та довіри до рішень ШІ в безпеці [17; 34; 126]. Усе вищезазначене зумовлює проблему побудови моделей IDS з використанням машинного навчання, які б були здатні працювати з високою точністю і водночас помірно вимогливими до обчислювальних ресурсів та могли використовуватися в реальному масштабі часу, або в режимі близькому до нього. Також існує проблема адаптації класифікаційних моделей до реального мережевого середовища та узагальнення таких моделей. Разом з тим важливою є необхідність побудови ізольованих та контрольованих середовищ, в яких можна проводити дослідження IDS.

РОЗДІЛ 2
МЕТОДИ ТА МОДЕЛІ ПРИЙНЯТТЯ РІШЕНЬ У ДВОРІВНЕВІЙ СИСТЕМІ ВИЯВЛЕННЯ ВТОРГНЕНЬ

Системи виявлення вторгнень функціонують в умовах суттєвої асиметрії ризиків, за якої різні типи помилкових рішень мають принципово різні наслідки [21]. Пропуск атаки (false negative) може призвести до компрометації інформаційних ресурсів, тоді як помилкове спрацювання (false positive), як правило, означає лише додаткове навантаження на систему аналізу або оператора. У зв’язку з цим критерії ефективності IDS не можуть зводитися до мінімізації середньої помилки класифікації, а повинні враховувати відмінності у вартості помилкових рішень.

Зазначена асиметрія природно приводить до формалізації задачі виявлення вторгнень у межах теорії прийняття рішень, де оптимальність визначається мінімізацією очікуваних втрат [81; 84]. Така постановка дозволяє явно задати різну критичність помилок та пов’язати параметри класифікації з операційними вимогами безпеки. У цьому контексті результати роботи моделей розглядаються не як остаточні істини, а як підстави для ухвалення рішень з урахуванням допустимого рівня ризику.

Альтернативний підхід до формалізації протидії атакам наводиться у роботі С. Зибіна, В. Хорошка та ін. [73] та базується на ігрових моделях взаємодії між атакуючою та захисною сторонами, що дозволяє розглядати вибір стратегії захисту як задачу оптимізації виграшу.

Важливою особливістю аналізу мережевого трафіку є те, що різні етапи обробки відповідають різним типам ризику. Первинна детекція загроз орієнтована на своєчасне виявлення потенційно небезпечних подій, тоді як подальша інтерпретація спрямована на зменшення ризику хибної атрибуції їхнього характеру. Це зумовлює доцільність використання дворівневої архітектури прийняття рішень [27], у межах якої кожен етап оптимізується відповідно до власної ролі та цільових характеристик.

2.1. Наївний баєсівський класифікатор

Наївний баєсівський класифікатор (Naïve Bayes - NB) – це ймовірнісний алгоритм класифікації, що ґрунтується на теоремі Баєса з «наївним» припущенням про умовну незалежність ознак між собою (за умови відомого класу)[49]. Завдяки простоті реалізації та ефективності на практиці, NB набув широкого застосування у задачах класифікації і часто демонструє точність, порівнянну зі складнішими моделями [105; 117]. Нижче розглянуто математичні основи цього класифікатора, виведено його основні формули, проаналізовано варіанти моделі для дискретних і неперервних даних, а також обговорено статистичні переваги й недоліки.

NB використовує формулу Баєса для обчислення апостеріорної ймовірності класу C за наявності вектора ознак $X = (x_1, \dots, x_n)$:

$$P(C \mid X) = \frac{P(C)\,P(X|C)}{P(X)}, \tag{2.1}$$

де $P(C)$ – апріорна ймовірність класу (ймовірність появи класу без урахування даних), $P(X \mid C)$ – імовірність (правдоподібність) спостереження X за умови класу, а $P(C \mid X)$ – шуканий апостеріорний розподіл класу [49]. У задачі класифікації знаменник $P(X)$ є сталим для всіх класів (однакова норма для порівняння), тому рішення про клас приймають за максимізацією чисельника[49]. Зокрема, вибирають клас $\hat{c}$, що максимально збільшує добуток $P(C = c)P(X \mid C = c)$:

$$\hat{c} = \underset{c}{argmax}\ P(C = c)\ P(X \mid C = c). \tag{2.2}$$

Для обчислення $P(X \mid C = c)$ без додаткових припущень потрібно знати повний спільний розподіл всіх ознак $P(x_1, \dots, x_n \mid C = c)$. Кількість параметрів для його оцінювання зростає експоненційно зі збільшенням числа ознак, що робить безпосередню оцінку неможливою для великих n [102]. Наївний підхід полягає у різкому спрощенні цієї моделі через припущення умовної незалежності ознак: вважається, що всі компоненти X_i статистично незалежні один від одного в межах одного класу [49]. Формально це означає, що для будь-яких $i \neq j$ виконується $P(X_i, X_j \mid C) = P(X_i \mid C)\,P(X_j \mid C)$, а розширення на n вимірів дає факторизацію:

$$P(X_1, \dots, X_n \mid C = c) \;=\; \prod_{i=1}^{n} P\,(X_i \mid C = c) \qquad (2.3)$$

Це припущення різко зменшує число параметрів моделі, дозволяючи оцінювати лише розподіли окремих ознак при кожному класі замість повного спільного розподілу [102]. Підставивши факторизований розподіл у формулу Баєса, отримаємо правило класифікації наївного Баєса:

$$\hat{c} = \underset{c}{argmax}\; P(C = c)\; \prod_{i=1}^{n} P\,(X_i \mid C = c). \qquad (2.4)$$

Таким чином, рішення про клас здійснюється шляхом перемножування апріорної ймовірності класу на правдоподібності окремих ознак для цього класу.

Оцінювання параметрів. Для застосування NB потрібні оцінки $P(C = c)$ та умовних розподілів $P(\,X_i \mid C = c\,)$. Їх зазвичай отримують з навчальної вибірки методом максимального правдоподібності. Апріорні ймовірності класів оцінюються як частки прикладів відповідного класу: $\hat{P}(C = c) = N_c/N$ (де N_c – число навчальних прикладів класу c, N – загальна кількість прикладів). Умовні ймовірності дискретних ознак оцінюються як відносні частоти: $\hat{P}(\,X_i = x \mid C = c\,) = \frac{N_{c,i,x}}{N_c}$, де$N_{c,i,x}$ – число навчальних прикладів класу c з i-ою ознакою, значення якої дорівнює x.

Припущення незалежності ознак є ключовим спрощенням, що робить наївний Баєс практично застосовним, але водночас і основним джерелом похибки моделі. В реальних даних ознаки рідко бувають строго незалежними умовно щодо класу – на практиці існують кореляції та залежності між характеристиками об’єктів[117]. Тим не менш, NB часто демонструє несподівано високу точність навіть за порушення цього припущення[105; 117]. Емпірично виявлено, що на багатьох задачах (наприклад, класифікація текстів, медична діагностика тощо) простий наївний Баєс не поступається більш складним методам, таким як дерева рішень або нейронні мережі[105; 117]. Зокрема, порівняльні дослідження показали, що NB нерідко дає точність, співставну з алгоритмом C4.5 для побудови дерев рішень [117].

Теоретичні результати пояснюють цей феномен властивостями функції втрат «0-1». Для успішної класифікації не обов’язково точно оцінити всю багатовимірну ймовірність – достатньо правильно визначити, який клас має

максимум апостеріорної ймовірності [105]. Іншими словами, навіть якщо припущення незалежності спрощує реальний розподіл і призводить до похибок у значеннях $P(C \mid X)$, класифікатор все одно буде оптимальним (у сенсі мінімальної ймовірності помилки), якщо максимальний апостеріорний клас визначено правильно[105].

Втім, припущення умовної незалежності задає істотне обмеження гнучкості моделі. Якщо в даних наявні суттєві залежності між ознаками, які впливають на належність до класу, NB не зможе їх врахувати. Відомо, що при суто бінарних ознаках наївний Баєс реалізує лише лінійно роздільні гіперплощинами рішення і, відповідно, неспроможний правильно класифікувати дані, які нелінійно роздільні у просторі ознак [105]. Класичним прикладом є задача виключного АБО (XOR): дві бінарні ознаки мають значення лише в комбінації, але поодинці не визначають клас. NB в такому випадку дасть хибний результат, оскільки за відсутності індивідуально інформативних ознак він не може виявити їх спільний вплив [105]. Узагальненням XOR є так звані m-з-n концепції (логічні правила типу «істинно, якщо виконуються m з n умов»), для яких доведено, що наївний Баєс не є глобально оптимальним класифікатором [18; 105]. Отже, хоча припущення незалежності спрощує побудову моделі, воно визначає і межі її застосовності: наївний Баєс ефективний, коли або (і) ознаки дійсно майже незалежні, або (іі) залежності між ними не спотворюють відносної переваги правильного класу над іншими.

Модель наївного Баєса є породжувальною, тобто вимагає припущення про форму розподілу $P(X \mid C)$ для ознак. Залежно від природи даних використовують різні розподіли, що породжує кілька різновидів NB. У випадку дискретних ознак (категоріальні або бінарні змінні) найбільш поширені такі моделі:

Для неперервних ознак найпопулярнішим є припущення про нормальний розподіл (Гаусів розподіл). Гаусівський наївний Баєс (Gaussian NB) вважає, що кожна ознака X_i при належності об’єкта до класу c розподілена нормально з певними середнім μ_{ci} та дисперсією σ_{ci}^2, які характеризують цей клас. Тоді умовна щільність ймовірності має вигляд:

$$P(X_i = x_i \mid C = c) = \frac{1}{\sqrt{2\pi\,\sigma_{ci}^2}}\, exp\left(-\frac{(x_i-\mu_{ci})^2}{2\sigma_{ci}^2}\right). \quad (2.5)$$

Оцінки μ_{ci} та σ_{ci} обчислюють із навчальних даних як вибіркове середнє та дисперсію відповідної ознаки для класу c. Гаусівський NB фактично є окремим випадком моделі нормального дискримінантного аналізу з діагональною коваріаційною матрицею (тобто із припущенням нульової кореляції ознак). Якщо припустити однакову дисперсію $\sigma_{ci}^2 = \sigma_i^2$ для всіх класів, то граничні поверхні, що відділяють класи, будуть лінійними (подібно до лінійного дискримінантного аналізу). У загальному випадку різних дисперсій класи розділяються квадратичними поверхнями (квадратичний дискримінант). Незважаючи на ці спрощення, Gaussian NB часто достатньо точно моделює реальні дані, особливо коли розподіл ознак близький до нормального або можлива апроксимація його сумішшю нормальних розподілів.

Наївний баєсівський класифікатор володіє низкою суттєвих переваг з погляду статистики та практичного використання, наприклад простота та ефективність, висока точність тощо. Розглянемо детальніше його переваги.

NB надзвичайно простий у реалізації та обчислювально швидкий. Навчання зводиться до підрахунку частот або обчислення простих статистик (середніх і дисперсій), а класифікація – до кількох добутків (або сум у логарифмічному просторі). Число параметрів зростає лінійно з кількістю ознак, на відміну від експоненційного зростання в загальному випадку без незалежності [102]. Така параметрична простота означає, що для навчання потрібно менше даних, і модель стійкіша до перенавчання (низька дисперсія оцінок). NB добре працює навіть на високорозмірних просторах ознак (тисячі і більше), де інші моделі потребували б складних методів відбору ознак чи нормалізації.

Попри «наївність» припущень, NB часто забезпечує точність, наближену до точності більш складних моделей. Експериментально показано, що на багатьох стандартних наборах даних (UCI, текстові корпуси тощо) наївний Баєс не поступається алгоритмам рішення дерев, найближчих сусідів і навіть подекуди конкурує з методами глибокого навчання для достатньо простих задач [105; 117]. Його успіх частково пояснюється тим, що мінімізація 0-1 втрат вимагає

правильно знаходити лише найбільш ймовірний клас, а не точні ймовірності всіх класів [105]. NB є статистично несуперечливим (consistent) класифікатором: якщо справжнє розподілення ознак дійсно задовольняє незалежність, то за нескінченного обсягу даних NB наближається до баєсівського оптимуму (найменшої можливої помилки). Навіть коли це припущення не виконується, NB може залишатися близьким до оптимального, якщо порушення не змінюють того, який клас є найімовірнішим [105]. Іншими словами, NB має високий біас (похибка апроксимації через спрощену модель), але дуже низьку дисперсію (стабільність оцінок з вибірки до вибірки). Це робить його корисним у ситуаціях, коли складніші моделі можуть перенавчитись. Також цей метод має високу ефективність для задач двійкової класифікації [38].

Завдяки небагатьом параметрам NB не потребує дуже великих навчальних вибірок для надійної оцінки ймовірностей. Більш того, включення нерелевантних або слабо інформативних ознак не сильно шкодить класифікації – якщо ознака статистично незалежна від мітки класу, вона фактично не вплине на рішення, оскільки ймовірності $P(X_i \mid C)$ будуть приблизно однакові для всіх класів і матимуть мінімальний вплив на добуток. Таким чином, NB є доволі стійким до присутності шумових ознак: зайві ознаки просто додають множники, близькі до константи, і не «збивають» вибір класу. З цієї причини NB добре масштабується до задач з великою кількістю потенційних факторів (наприклад, аналіз текстів з дуже великим словником ознак).

Наївний Баєс – генеративна модель, що дає повний розподіл $P(C, X)$. Це дозволяє природно обробляти приклади з частково пропущеними ознаками (через маргіналізацію останніх) та легко інтегрувати апріорні знання через відповідні розподіли. Модель можна інкрементно оновлювати по мірі надходження нових даних, просто коригуючи підрахунки частот. Деякі реалізації, як-от у бібліотеці scikit-learn, підтримують поступове донавчання NB на потоках даних [49]. Такі властивості є корисними для динамічних систем, що вчаться на нових зразках без повного перенавчання з нуля.

Поряд із сильними сторонами, наївний Баєсівський класифікатор має і важливі недоліки, що випливають із його спрощень, наприклад, такі як нереалістичність припущення про незалежність ознак.

У більшості реальних задач ознаки мають певний ступінь кореляції, і повне ігнорування цих залежностей призводить до спрощеної моделі. Якщо дві чи більше ознаки разом несуть інформацію про клас, яку не передають поодинці, NB цю інформацію втратить. Такі взаємодії (наприклад, комбінації симптомів у медицині чи спільна поява слів у тексті) не можуть бути враховані моделлю з незалежними правдоподібностями. Це може знижувати точність у випадках, де залежності відіграють вирішальну роль. Як зазначалось, NB неспроможний коректно класифікувати дані, які не лінійно роздільні у просторі ознак, зокрема XOR-подібні концепції [105]. В загальному, якщо припущення незалежності серйозно порушене, модель має систематичну похибку (bias) і поступається методам, що можуть враховувати взаємодії між ознаками (наприклад, дерева рішень, випадкові ліси, логістична регресія з поліноміальними ознаками тощо).

Також Наївний Баєс схильний давати погані оцінки самих апостеріорних ймовірностей. Хоча вибір класу може бути правильним, розраховані значення $P(C \mid X)$ часто зміщені через сильні спрощення. Дослідження показали, що NB погано калібрує ймовірності: він може впевнено давати $P(\text{клас}|X) \approx 0.99$ там, де реальна ймовірність далека від 99%, або навпаки [117]. Це особливо помітно, коли модель застосовувати не для класифікації (0-1 втрати), а для задач регресії чи прогнозування ймовірнісних величин – у таких випадках наївний Баєс проявляє значно гірші результати [117]. Причина в тому, що залежні ознаки фактично «дублюють» внесок у добутку ймовірностей, що приводить до переконливо екстремальних (але хибних) значень постеріорів. Тому у сферах, де важлива саме якість прогнозованих ймовірностей (наприклад, оцінювання ризиків), NB менш придатний без додаткового калібрування (наприклад, методом ізотонічної регресії).

Якщо якась комбінація ознака-клас взагалі не траплялась у навчальних даних, наївний класифікатор присвоїть їй нульову ймовірність. Це означає, що за наявності будь-якої нової (раніше не баченого значення ознаки або категорії)

модель відразу буде на 100% впевнена, що відповідний клас неможливий (оскільки добуток міститиме нуль). В реальності ж поодиноке неспостереження не означає неможливість події. Для подолання цієї проблеми й застосовується згладжування (наприклад, додавання одиниці до лічильників – лапласівське згладжування) [49]. Однак вибір коефіцієнта згладжування $\boldsymbol{\alpha}$ впливає на результати: надмірне згладжування може зменшити чутливість до реальних частот, а недостатнє – залишити ризик нульових ймовірностей. Це тонке місце в налаштуванні NB, особливо для даних з рідкісними категоріями ознак.

Як було згадано, NB з бінарними ознаками реалізує лінійний розділ класи, що обмежує його виразну здатність [105]. Для розв’язання задач з нелінійними межами між класами сам по собі NB непридатний. Існують узагальнення, що додають залежності між ознаками, наприклад, так звані деревоподібні баєсівські класифікатори (Tree Augmented Naive Bayes, TAN), де структура моделі допускає деякі зв’язки між ознаками. Вони покращують точність за рахунок ускладнення моделі, частково усуваючи «наївність». Проте такі модифікації виходять за рамки класичного NB і потребують додаткових алгоритмів навчання структури мережі, що збільшує обчислювальні витрати.

Підсумовуючи, наївний Баєс – це модель із високою упередженістю і низькою дисперсією. Він робить сильні припущення, які не завжди точні, проте саме ці припущення дають йому переваги у швидкості та стабільності. У багатьох ситуаціях (особливо за великої розмірності ознак і відносно невеликих вибірок) такий компроміс виявляється виправданим, і NB забезпечує конкурентоспроможну точність.

Разом із тим наївний баєсівський класифікатор знайшов застосування у різних напрямках, де потрібна автоматична класифікація чи фільтрація інформації. Він став особливо популярним у задачах обробки тексту та інформаційного пошуку. Зокрема, NB давно і успішно використовується для класифікації документів за темами, фільтрації спаму в електронній пошті, аналізу тональності текстів (визначення сентименту), класифікації новин та інших контент-модераційних завдань[49; 105]. У біомедичних застосуваннях наївний Баєс слугує простим але ефективним інструментом для медичної діагностики та

прогнозування захворювань на основі сукупності симптомів і лабораторних показників [105]. Його використовували у системах підтримки рішення лікаря, де важлива інтерпретованість і швидкість – NB надає зрозумілий ймовірністий висновок на основі наявних симптомів.

Окремо варто відзначити застосування NB у галузі кібербезпеки. Завдяки швидкості та невибагливості до ресурсів, NB інтегрується в інтелектуальні системи виявлення атак та аномалій в мережевому трафіку. Зокрема, його використовують в системах виявлення вторгнень (IDS) для класифікації мережевих з'єднань як нормальні або зловмисні за сукупністю ознак трафіку [105]. Наївний Баєс успішно моделює нормальну поведінку мережі та виявляє відхилення, що можуть свідчити про кібератаки, при цьому забезпечуючи низьку обчислювальну вартість і можливість навчання на потокових даних. Він також застосовується для фільтрації трафіку – наприклад, для виявлення спаму в мережевих пакетах, шкідливих URL чи ненормативного контенту, де задача зводиться до класифікації об'єктів (пакетів, повідомлень) на допустимі чи заборонені.

Таким чином, наївний баєсівський класифікатор є одним із базових інструментом у багатьох сферах, що вимагають автоматичного ухвалення рішень на основі даних. Його теоретична простота поєднується з несподівано високою ефективністю, завдяки чому NB і донині залишається важливою складовою арсеналу машинного навчання – як самостійно, так і у складі складніших інтелектуальних систем виявлення та аналізу мережевих атак.

2.2. Вибір та роль алгоритму Random Forest у дворівневій моделі

Random Forest (метод випадкових лісів) – це ансамблевий метод машинного навчання, який будує велику кількість незалежних дерев рішень та агрегує їхні передбачення для отримання кінцевого результату. Формально, Random Forest складається з колекції T дерев-регресорів або класифікаторів (рис 2.1) $h(x, \Theta_k),\ k = 1, \dots, T$, де Θ_k – незалежні випадкові вектори, що визначають процес побудови k-го дерева [10]. Кожне дерево голосує за певний

клас, а прогноз ансамблю визначається як голосування більшості серед усіх дерев (для класифікації) або середнє значення прогнозів (для регресії) [9]. Таким чином, Random Forest поєднує результати багатьох «слабких» моделей-дерев в одну сильнішу модель.

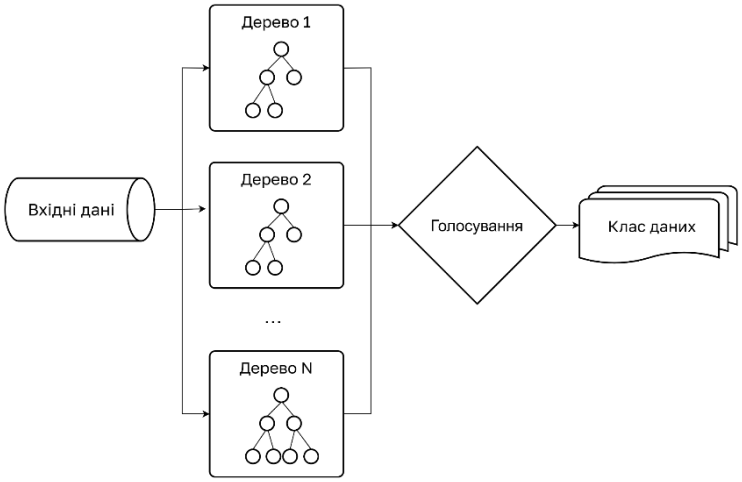

Рис. 2.1. Принцип роботи методу Random Forest

Джерело: сформовано автором за [10]

Метод Random Forest використовує два ключові механізми рандомізації: беггінг (bootstrap aggregating) і випадковий вибір підмножини ознак. По-перше, для тренування кожного дерева використовується свій бутстрап-вибірковий набір даних – випадкова вибірка з навчального набору з поверненням (тобто деякі приклади можуть повторюватися, а близько 36% прикладів залишаються невідібраними) [10]. По-друге, при розбитті кожного вузла дерева алгоритм випадково обирає підмножину ознак (випадковий підпростір ознак) і шукає найкраще розбиття лише серед них, що зменшує кореляцію між окремими деревами [56; 115]. Завдяки цьому дерева в лісі менш схильні робити однакові помилки, і ансамбль стає більш стійким до перенавчання. Після побудови великої кількості дерев, їхні відповіді об’єднуються: у випадку класифікації обирається клас, за який проголосувала більшість дерев (мажоритарне голосування), а в випадку регресії – усереднюється числовий прогноз [9]. Така схема забезпечує високу точність моделі та її стійкість до шумів у даних [10].

Якість побудованої моделі, її здатність до узагальнення та ефективність у виявленні складних шаблонів у даних значною мірою залежать від вибору гіперпараметрів [101].

Кількість дерев у ансамблі (n_estimators) – цей параметр визначає кількість дерев рішень, що формують ансамбль. Зі збільшенням кількості дерев підвищується стабільність моделі, знижується дисперсія передбачень, а ризик перенавчання зменшується, хоча обчислювальні витрати при цьому зростають. У своїй фундаментальній роботі Breiman [10] доводить, що за великої кількості дерев узагальнююча похибка Random Forest стабілізується.

Формально, остаточне передбачення для прикладу $\boldsymbol{x}$визначається як:

$$\hat{y}(x) = \arg\max_{y} \sum_{t=1}^{T} \mathbb{1}\{h_t(x) = y\} \tag{2.6}$$

де $T = n_{\text{estimators}}$, а $h_t(x)$ – передбачення t-го дерева, $\mathbb{1}$ – індикаторна функція.

Максимальна глибина дерева (max_depth) – цей параметр обмежує кількість рівнів у дереві рішень. Встановлення верхньої межі дозволяє уникати перенавчання, особливо в задачах з обмеженим обсягом навчальних даних або наявним шумом. [26], підкреслюють, що обмеження глибини зменшує варіативність дерев і запобігає утворенню занадто складних розгалужень.

Кількість вузлів у повному бінарному дереві залежить від глибини $N = 2^d - 1$, де d– глибина дерева.

Мінімальна кількість зразків для поділу (min_samples_split) – цей гіперпараметр задає мінімальну кількість зразків, які повинні бути у вузлі, щоб дозволити його поділ на підвузли. Значення min_samples_split безпосередньо впливає на форму дерева: більше значення обмежує глибину і підвищує узагальнюючу здатність, менше – сприяє гнучкості, але може призвести до переобучення [120].

Мінімальна кількість зразків у листовому вузлі (min_samples_leaf) – цей параметр задає мінімальну кількість зразків, які можуть опинитись у листі. Він важливий для забезпечення стабільності передбачень, особливо в задачах з класовим дисбалансом. [26] зазначають, що фіксація значення min_samples_leaf на рівні >1 знижує дисперсію моделі.

Формально, розбиття вузла можливе лише за умови:

$$\begin{cases} n_{\text{left}} \geq \text{min\_samples\_leaf} \\ n_{\text{right}} \geq \text{min\_samples\_leaf} \end{cases} \tag{2.7}$$

Кількість ознак, доступних при кожному поділі (max_features) – цей параметр визначає розмір підмножини ознак, яка розглядається при побудові кожного вузла дерева. Значення max_features контролює ступінь випадковості в ансамблі: чим менше ознак розглядається, тим менш корельованими будуть дерева, що знижує дисперсію моделі [10].

Можливі значення цього параметра: «sqrt» ($\sqrt{p}$) або «log2» ($log_2 p$) , де p – загальна кількість ознак.

Використання вибірки з поверненням (bootstrap). Якщо параметр bootstrap = True, то для навчання кожного дерева вибирається випадкова вибірка з поверненням. За оцінками, близько 63.2% унікальних прикладів потрапляють у кожну bootstrap-вибірку [106].

Ймовірність того, що деякий приклад x_iне потрапить до вибірки:

$$P(x_i \notin S) = \left(1 - \frac{1}{n}\right)^n \overset{n\to\infty}{\to} \frac{1}{e} \approx 0.368 \quad (2.8)$$

Функція оцінки якості розбиття (criterion). Random Forest використовує одну з двох основних функцій для вимірювання «неоднорідності» у вузлі. Так Gini $G = 1 - \sum_{i=1}^{K} p_i^2$ використовується за замовчуванням у класифікації, де p_i – частка класу i у вузлі, K– кількість класів. Обидві функції мінімізуються при найбільш «чистому» поділі, в такому випадку значення оцінок дорівнюють нулю, що відповідає наявності об'єктів лише одного класу. Вибір функції може впливати на стабільність дерева та глибину поділу, однак відмінності в результатах при великій кількості дерев зазвичай незначні [10].

Вибір алгоритму класифікації для другого рівня інтелектуальної системи виявлення вторгнень є принциповим етапом проєктування та не може зводитися до використання популярних або формально ефективних методів. У наукових дослідженнях з машинного навчання вибір моделі завжди ґрунтується на балансі між кількома взаємопов’язаними критеріями, зокрема точністю класифікації, обчислювальною ефективністю, стійкістю до шуму, здатністю до узагальнення та придатністю до роботи в умовах відкритого набору класів (open-set recognition).

У межах запропонованої дворівневої архітектури системи виявлення вторгнень алгоритм Naïve Bayes використовується на першому рівні як швидкий фільтр для грубого відокремлення потенційно небезпечного трафіку. Другий рівень призначений для поглибленого аналізу відібраних мережевих потоків і потребує більш потужного, стабільного та інтерпретованого класифікатора. Для реалізації цього рівня обрано алгоритм Random Forest.

З метою обґрунтування такого вибору доцільно порівняти Random Forest з альтернативними підходами машинного навчання, які теоретично можуть бути застосовані для задачі виявлення аномалій у мережевому трафіку, але мають суттєві обмеження саме в розглянутому сценарії.

Методи на основі найближчих сусідів, зокрема k-NN та його відкриті модифікації типу Open-Set Nearest Neighbor [87], мають привабливі властивості для задач open-set recognition. Використання співвідношення відстаней до найближчих сусідів різних класів дозволяє виявляти зразки, що не належать до жодного з відомих класів. Водночас k-NN належить до «лінивих» алгоритмів [116] і не формує внутрішньої моделі, а зберігає всю навчальну вибірку, що зумовлює надзвичайно високі обчислювальні витрати на етапі прогнозування [116]. За великої кількості ознак та мільйонів записів мережевого трафіку такий підхід практично унеможливлює роботу системи в режимі реального часу.
На відміну від цього, Random Forest формує компактну модель у вигляді ансамблю дерев і не потребує зберігання всієї навчальної вибірки. Це забезпечує значно вищу швидкість прогнозування та передбачувану обчислювальну складність, що є критично важливим для IDS. Крім того, на відміну від k-NN, ефективність Random Forest суттєво менше залежить від високої розмірності простору ознак, оскільки випадковий підбір ознак у вузлах дерев знижує негативний вплив великої розмірності даних.

Метод опорних векторів та його відкриті модифікації мають ґрунтовну теоретичну базу для задач відкритого розпізнавання. Підходи на кшталт 1-vs-Set Machine SVM дозволяють формально обмежувати область прийняття рішень і знижувати ризик помилкової класифікації віддалених зразків [91]. Водночас

практичне застосування SVM у задачах аналізу мережевого трафіку великих обсягів супроводжується високою чутливістю до гіперпараметрів [113] і значними обчислювальними витратами, що зростають щонайменше квадратично зі збільшенням кількості навчальних зразків [62]. У порівнянні з цим Random Forest характеризується значно простішим налаштуванням, стійкістю до вибору гіперпараметрів та лінійно масштабованою процедурою навчання. Це робить його придатним для роботи з великими датасетами мережевого трафіку та багатокласових сценаріїв IDS без суттєвого ускладнення процесу навчання.

Алгоритми градієнтного бустингу, такі як XGBoost, LightGBM або CatBoost, часто демонструють дуже високу точність на табличних даних [123] і активно використовуються в прикладних задачах класифікації. Окремі реалізації підтримують механізми оцінки невизначеності прогнозу, що теоретично дозволяє виявляти зразки, які виходять за межі навчального розподілу. Водночас бустингові ансамблі є чутливими до шуму [103] та аномальних викидів, що є характерним для реального мережевого трафіку, особливо зібраного в емульованих середовищах.

На відміну від послідовних бустингових моделей, Random Forest використовує механізм bagging, у якому дерева будуються незалежно. Це забезпечує вищу стійкість до шуму та запобігає накопиченню помилок на випадкових флуктуаціях даних. Крім того, паралельна побудова дерев у Random Forest дозволяє ефективніше використовувати обчислювальні ресурси та зменшувати час обробки великих потоків мережевого трафіку.

Глибокі нейронні мережі здатні автоматично формувати інформативні ознаки та мають спеціалізовані розширення для задач відкритого розпізнавання, зокрема підхід OpenMax [87; 119]. Проте такі моделі залишаються слабкоінтерпретованими та схильними до надмірної впевненості у своїх прогнозах, що є критичним недоліком у задачах кібербезпеки.
У цьому контексті Random Forest має суттєву перевагу завдяки вищому рівню інтерпретованості. Аналіз важливості ознак дозволяє пояснювати рішення моделі та ідентифікувати характеристики мережевого трафіку, що найбільше

вплинули на класифікацію. Крім того, механізм голосування дерев дає змогу контролювати рівень впевненості прогнозу і відхиляти нетипові зразки.

На тлі розглянутих альтернатив Random Forest є ансамблевим методом, який поєднує прийнятну точність із високою стійкістю до шуму. Завдяки використанню механізму bagging кожне дерево навчається на випадковій підмножині даних і ознак, що зменшує ризик перенавчання та забезпечує стабільну роботу моделі на зашумлених і гетерогенних даних мережевого трафіку. Алгоритм добре масштабується на великі набори даних і забезпечує високу швидкість прогнозування, оскільки класифікація нового зразка зводиться до виконання обмеженої кількості логічних перевірок у кожному дереві та агрегації результатів голосування. Незалежність дерев дозволяє ефективно паралелізувати як навчання, так і застосування моделі [10].

Важливою перевагою Random Forest є можливість інженерно прозорої реалізації механізму відкритого набору класів. Використання порогу довіри на основі частки голосів дерев або апостеріорних ймовірностей дозволяє відхиляти зразки, для яких модель не має достатньої впевненості, без модифікації архітектури алгоритму. Крім того, Random Forest забезпечує прийнятний рівень інтерпретованості: аналіз важливості ознак дозволяє визначити, які характеристики даних мають найбільший вплив на рішення моделі [67].

Водночас Random Forest має і низку обмежень. Механізм open-set recognition у цьому алгоритмі є евристичним і не супроводжується строгими теоретичними гарантіями мінімізації ризику відкритого простору. Як і інші деревоподібні моделі, він має обмежену здатність до екстраполяції за межі простору ознак, представленого в навчальній вибірці [124]. Зі збільшенням кількості дерев і глибини ансамблю зростають вимоги до пам’яті, а інтерпретованість окремих рішень має агрегований характер і не завжди дозволяє однозначно пояснити класифікацію конкретного зразка.

Узагальнюючи, використання Random Forest як другого рівня IDS-системи є обґрунтованим з позицій практичного застосування. Алгоритм не орієнтований на забезпечення формальних гарантій open-set recognition або максимізацію точності на ідеалізованих вибірках, однак характеризується високою

стабільністю, масштабованістю та стійкістю до шуму, що є критично важливим для аналізу реального мережевого трафіку. [11]. Саме поєднання цих властивостей робить Random Forest доцільним вибором для побудови надійної та інтерпретованої моделі для використання системи виявлення вторгнень.

2.3 Теоретична модель дворівневої системи прийняття рішень у IDS

Ефективність системи виявлення вторгнень значною мірою визначається не лише вибором конкретних алгоритмів машинного навчання, а й організацією процесу прийняття рішень. У реальних мережевих середовищах задача виявлення атак характеризується високою асиметрією ризиків, змінністю трафіку та наявністю ситуацій, у яких повна та однозначна класифікація є неможливою або економічно недоцільною. За цих умов використання єдиного класифікаційного етапу, який одночасно відповідає і за детекцію атаки, і за її інтерпретацію, є методично обмеженим. [57]

Запропонована у роботі дворівнева архітектура прийняття рішень ґрунтується на розділенні функцій класифікації між двома послідовними етапами, кожен з яких оптимізується з урахуванням власного типу ризику та операційної ролі в IDS. [27]

Перший рівень архітектури виконує функцію первинної детекції та орієнтований на розв’язання бінарної задачі «нормативний трафік / потенційна атака». Основним завданням цього рівня є швидке відсікання явно безпечного трафіку та передавання підозрілих потоків на подальший аналіз. Ключовим типом ризику на цьому етапі є пропуск атаки, який у контексті інформаційної безпеки має критичні наслідки. Відповідно, перший рівень налаштовується на підвищену чутливість і допускає збільшення кількості помилкових спрацювань як прийнятний компроміс заради мінімізації ризику false negative.

Другий рівень архітектури призначений для інтерпретації та атрибуції типу атаки серед множини відомих класів і застосовується лише до трафіку, який був відібраний першим рівнем як підозрілий. На цьому етапі домінуючим стає інший тип ризику – хибна атрибуція типу загрози, яка може призвести до неправильних аналітичних висновків або неадекватних дій реагування. У зв’язку з цим другий

рівень орієнтований не на максимальну чутливість, а на надійність і стабільність інтерпретації результатів [93].

У випадках, коли наявної інформації недостатньо для надійної інтерпретації результату, система повинна мати можливість утриматися від остаточного рішення, що буде формалізовано на другому рівні класифікації.

Таким чином, дворівнева архітектура реалізує послідовне уточнення гіпотези щодо стану мережевого трафіку. Перший рівень відповідає за своєчасне виявлення потенційної загрози, тоді як другий рівень забезпечує коректну інтерпретацію її характеру з урахуванням рівня впевненості моделі. Розділення цих функцій дозволяє застосовувати різні критерії оптимальності на різних етапах аналізу та формувати керовану політику прийняття рішень, адаптовану до практичних умов експлуатації системи виявлення вторгнень [78].

Обґрунтування порогу рішення першого рівня (бінарна класифікація)

Для формалізації порогового правила першого рівня використаємо баєсівську постановку задачі прийняття рішень із асиметричними втратами [21].

$$L_b = \begin{pmatrix} L_{(norm,norm)} & L_{(norm,threat)} \\ L_{(attack,norm)} & L_{(attack,threat)} \end{pmatrix} = \begin{pmatrix} 0 & C_{FP}^b \\ C_{FN}^b & 0 \end{pmatrix} \quad (2.9)$$

Де C_{FN}^b – вартість помилки типу *false negative* (атака розпізнана як нормальний трафік), C_{FP}^b – вартість помилки типу *false positive* (нормальний трафік розпізнаний як атака).

Така структура матриці втрат відображає різну критичність помилок першого рівня IDS [71]. Нехай метод Naïve Bayes як результат класифікації повертає вектор апостеріорних властивостей

$$\pi(x) = \begin{pmatrix} P_{(norm|x)} \\ P_{(attack|x)} \end{pmatrix} = \begin{pmatrix} 1-p \\ p \end{pmatrix}, \text{ де } p = P(attack|x), \quad (2.10)$$

Таке представлення дозволяє звести задачу прийняття рішення до аналізу єдиної скалярної величини p, що є зручним з точки зору реалізації та подальшої інтерпретації.

Позначимо через $d \in \{norm,threat\}$ можливе рішення першого рівня. Умовний ризик цього рішення за фіксованого спостереження x визначається стандартним чином: $R_b(d|x) = \sum_y L_b(y,d)\, P(y|x)$.

Опишемо детальніше ризики обох можливих рішень для першого рівня моделі. Для рішення *norm* (визнання трафіку нормативним):

$$R_b(norm|x) = \begin{pmatrix} 0 & C_{FN}^b \end{pmatrix} \begin{pmatrix} 1-p \\ p \end{pmatrix} = 0 \cdot (1 - p) + C_{FN}^b \cdot p = C_{FN}^b \cdot p; \quad (2.11)$$

Цей вираз відображає той факт, що ризик рішення *norm* зростає пропорційно апостеріорній імовірності атаки.

Для рішення *threat* (визнання трафіку підозрілим):

$$R_b(threat|x) = \begin{pmatrix} C_{FP}^b & 0 \end{pmatrix} \begin{pmatrix} 1-p \\ p \end{pmatrix} = C_{FP}^b(1 - p); \quad (2.12)$$

У цьому випадку ризик зумовлений ймовірністю того, що трафік є насправді нормативним.

Згідно з баєсівським критерієм оптимальності, оптимальне рішення першого рівня визначається як

$$f_b(x) = arg \min_{d \in \{norm,threat\}} R_b(d|x) \quad (2.13)$$

З урахуванням того, що для IDS пропуск атаки є більш критичним, рішення *threat* повинно прийматись у всіх випадках, коли його очікуваний ризик не перевищує ризик рішення *norm* [21]: $R_b(threat|x) \leq R_b(norm|x)$.

Підставимо в нерівність значення вказані у формулі та розв'язуємо нерівність

$$C_{FP}^b(1-p) \leq C_{FN}^b \quad (2.14)$$

$$p \geq \frac{C_{FP}^b}{C_{FN}^b + C_{FP}^b} \quad (2.15)$$

Отриманий вираз задає аналітично обґрунтований поріг [19; 71]:

$$\tau_b = \frac{C_{FP}^b}{C_{FN}^b + C_{FP}^b} \quad (2.16)$$

який визначає правило прийняття рішень на першому рівні дворівневої IDS. Таким чином, трафік визнається підозрілим, якщо апостеріорна ймовірність атаки перевищує цей поріг; в іншому випадку він класифікується як нормативний.

Хоча порогове правило першого рівня може бути безпосередньо сформульоване у просторі апостеріорних імовірностей, на практиці доцільно виконувати прийняття рішень у просторі log-odds – логарифма відношення

ймовірностей альтернативних гіпотез. Використання log-odds дозволяє інтерпретувати поріг першого рівня як міру переваги гіпотези атаки над нормативною поведінкою, що є зручним для подальшої калібрації в експериментальній частині. Для бінарної задачі класифікації визначимо log-odds (логарифм відношення шансів) як:

$$\ell(x) = log\frac{P(\text{attack}|x)}{P(\text{norm}|x)} = log\frac{p}{1-p}. \quad (2.17)$$

Перепишемо умову у термінах відношення імовірностей:

$$\frac{p}{1-p} \geq \frac{C_{FP}^b}{C_{FN}^b}. \quad (2.18)$$

Застосувавши логарифмування обох частин нерівності, отримаємо еквівалентне правило:

$$\log \frac{p}{1-p} \geq \log \frac{C_{FP}^b}{C_{FN}^b}. \quad (2.19)$$

Позначимо аналітичний поріг у просторі log-odds як:

$$\tau_b = log\frac{C_{FP}^b}{C_{FN}^b}. \quad (2.20)$$

Співвідношення C_{FN}^b/C_{FP}^bу цій роботі розглядається не як фіксована константа, а як параметр політики IDS, що відображає допустимий рівень помилкових спрацьовувань при заданому рівні ризику пропуску атак.

Тоді правило прийняття рішення на першому рівні можна записати у компактному вигляді:

$$f_b(x) = \begin{cases} \text{threat}, & \ell(x) \geq \tau_b, \\ \text{norm}, & \ell(x) < \tau_b. \end{cases} \quad (2.21)$$

Обґрунтування порогу рішення другого рівня (класифікація з відмовою)

Другий етап дворівневої системи виявлення вторгнень призначений для багатокласової атрибуції типу атаки серед множини відомих класів $\mathcal{K} = \{1,2,\dots,K\}$

із можливістю відмови від прийняття рішення у випадках, коли наявної інформації недостатньо для надійної атрибуції. Відповідно, простір можливих рішень визначається як $d \in \mathcal{D} = \mathcal{K} \cup \{\text{unknown}\}$.

Такий підхід відповідає практичним вимогам IDS, де примусова класифікація у ситуації невизначеності може бути більш шкідливою, ніж контрольована відмова від рішення .

Як класифікатор другого рівня використовується алгоритм Random Forest, який для кожного об'єкта xформує нормалізований вектор оцінок класів $\hat{\mathrm{p}}(x) = (\hat{p}_1(x), \hat{p}_2(x), \dots, \hat{p}_K(x)), \sum_{k=1}^{K} \hat{p}_k\,(x) = 1$.

У подальшому вектори $\hat{p}_k(x)$ розглядаються як нормалізовані оцінки впевненості класифікатора, що мають імовірнісну інтерпретацію лише в сенсі відносних часток або ваг, але не є строго каліброваними апостеріорними ймовірностями.

Для подальшого аналізу введемо позначення класу з максимальною оцінкою та відповідного значення:

$$k^*(x) = \arg\max_{k\in\mathcal{K}} \hat{p}_k(x), p_{\max}(x) = \max_{k\in\mathcal{K}} \hat{p}_k(x). \qquad (2.22)$$

Величина $p_{\max}(x)$відображає рівень консенсусу ансамблю щодо найбільш імовірного типу атаки.

Для формалізації механізму відмови від рішення застосуємо спрощену модель втрат, яка відповідає специфіці задачі атрибуції типу атаки в IDS. Розглядаються такі випадки:

- правильна атрибуція типу атаки – нульові втрати;
- хибна атрибуція типу атаки – втрати C_{MC}.
- відмова від рішення (*unknown*) – втрати C_r.

Припустимо, що параметри моделі задовольняють умову $0 < C_r < C_{MC}$.

Зазначене співвідношення відповідає принципу контрольованої відмови, сформульованому на початку підрозділу.

Функцію втрат запишемо у вигляді:

$$L(y,d) = \begin{cases} 0, & d = y, \\ C_{MC}, & d \in \mathcal{K}, d \neq y, \\ C_r, & d = unknown \end{cases} \quad (2.23)$$

Умовний ризик прийняття рішення dза для об'єкта xвизначається як математичне сподівання втрат з урахуванням оцінок впевненості моделі:

$$R(d \mid x) = \sum_{j=1}^{K} L(j,d)\ \hat{p}_j(x). \quad (2.24)$$

Розглянемо ризики для різних типів рішень.

Для рішення про атрибуцію конкретного класу $k \in \mathcal{K}$ правильна класифікація не призводить до втрат, тоді як будь-який інший істинний клас означає хибну атрибуцію з вартістю C_{MC}. Таким чином,

$$R(k \mid x) = \sum_{j \neq k} C_{MC}\ \hat{p}_j(x) = C_{MC}(1 - \hat{p}_k(x)). \quad (2.25)$$

Цей вираз показує, що ризик атрибуції зменшується зі зростанням впевненості моделі у відповідному класі.

Ризик відмови від рішення не залежить від розподілу оцінок класів, оскільки відмова має сталу вартість:

$$R(\text{unknown} \mid x) = \sum_{j=1}^{K} C_r\ \hat{p}_j(x) = C_r \quad (2.26)$$

Аналогічно до першого рівня, рішення другого рівня обирається за критерієм мінімізації умовного ризику з урахуванням втрат атрибуції та відмови.

$$f_{rf}(x) = \arg\min_{d \in \mathcal{D}} R(d \mid x) \quad (2.27)$$

Атрибуція класу kє доцільною тоді і лише тоді, коли її очікуваний ризик не перевищує ризик відмови від рішення $R(k \mid x) \leq R(\text{unknown} \mid x)$.

Підставивши відповідні вирази для ризиків, отримаємо нерівність:

$$C_{MC}(1 - \hat{p}_k(x)) \leq C_r \quad (2.28)$$

з якої випливає порогова умова:

$$\hat{p}_k(x) \geq 1 - \frac{C_r}{C_{MC}} \quad (2.29)$$

Позначимо відповідний поріг прийняття рішення другого рівня як

$$\tau_{mc} = 1 - \frac{C_r}{C_{MC}} \quad (2.30)$$

Узагальнюючи отримані результати, оптимальне правило прийняття рішення на другому етапі класифікації має вигляд:

$$f_{rf}(\mathrm{x}) = \begin{cases} k^*(x), & p_{\max}(x) \geq \tau_{mc}, \\ \text{unknown}, & p_{\max}(x) < \tau_{\mathrm{mc}}. \end{cases} \quad (2.31)$$

Сформулюємо узагальнене правило прийняття рішення в запропонованій дворівневій моделі

$$f(x) = \begin{cases} norm, \ell(x) < \tau_b \\ k^*(x),\ \ell(x) < \tau_b \ \wedge p_{max} \geq \tau_{mc} \\ unknown,\ \text{інакше} \end{cases} \quad (2.32)$$

Таким чином, правило прийняття рішення у дворівневій IDS формалізується як задача вибору між автоматичною атрибуцією та відмовою від рішення на основі мінімізації очікуваного баєсівського ризику. Вартість відмови інтерпретується як операційні витрати, пов’язані з подальшою більш вартісною перевіркою трафіку, тоді як вартість хибної атрибуції відображає потенційні наслідки неправильного реагування на атаку. Запропоноване правило дозволяє аналітично визначити умови, за яких автоматична класифікація є доцільнішою за передачу об’єкта на більш ресурсоємний аналіз [122].

Запровадження порогових правил на першому та другому етапах класифікації формує єдину політику прийняття рішень у дворівневій системі виявлення вторгнень. Кожен із порогів виконує власну функцію та відповідає за контроль різних типів ризику, що унеможливлює їх трактування як незалежних або взаємозамінних параметрів.

Порогове правило першого рівня визначає умови передачі мережевого трафіку на подальший аналіз. Його налаштування орієнтоване на мінімізацію ризику пропуску атак і, відповідно, допускає підвищену кількість помилкових спрацювань. У межах загальної політики IDS цей поріг виконує роль фільтра з підвищеною чутливістю, який забезпечує своєчасне виявлення потенційно небезпечних подій.

Порогове правило другого рівня застосовується лише до трафіку, відібраного першим рівнем, і визначає умови атрибуції типу атаки або формування стану невизначеності. На цьому етапі основним є контроль ризику хибної інтерпретації результатів, тому поріг другого рівня налаштовується з

урахуванням рівня впевненості моделі та бажаного компромісу між повнотою атрибуції та кількістю відмов (unknown).

Таким чином, узгодження порогів визначає єдину політику прийняття рішень IDS, а не набір незалежних параметрів.

Узгоджена робота двох порогових механізмів реалізує послідовний процес прийняття рішень: від первинної детекції потенційної загрози до її інтерпретації з урахуванням рівня впевненості моделі. Такий підхід дозволяє чітко розділити відповідальність між етапами класифікації, підвищити інтерпретованість рішень і забезпечити керованість поведінки системи в умовах змінного мережевого середовища.

У процесі функціонування систем виявлення вторгнень мережевий трафік доцільно розглядати як стохастичний процес [36], статистичні характеристики якого можуть змінюватися в часі під впливом як легітимних факторів (зміна профілю користувачів, конфігурації мережі, інтенсивності сервісів), так і зловмисних дій. Такі зміни призводять до трансформації розподілів ознак або вихідних оцінок моделей машинного навчання. У науковій літературі подібні явища зазвичай описуються термінами *domain shift* або *concept drift*.

Традиційні підходи до оцінювання ефективності IDS, орієнтовані на статичні вибірки та *offline*-метрики, не забезпечують своєчасного виявлення подібних зсувів під час експлуатації системи. Внаслідок цього модель може формально зберігати працездатність, проте поступово втрачати узгодженість із реальними умовами мережевого середовища [25]. З огляду на це виникає необхідність у формалізованому механізмі контролю стабільності статистичних розподілів, який є незалежним від конкретної реалізації класифікатора та не потребує його перенавчання.

Перспективним підходом до розв’язання цієї задачі є використання ентропійних мір розбіжності, що дозволяють кількісно оцінювати відмінності між імовірнісними розподілами. На відміну від локальних показників якості класифікації, такі міри оперують узагальненими характеристиками розподілів і дають змогу виявляти системні зміни у структурі даних або вихідних рішень моделі.

2.4. Математична модель оцінювання зсуву домену мережевого трафіку на основі інформаційної дивергенції

Кардинальна проблема застосування методів машинного навчання (МН) в задачах виявлення вторгнень – це порушення гіпотези про незалежність та ідентичність розподілу даних під час переходу від етапу навчання до етапу експлуатації. Параметри мережевого трафіку у цільовому середовищі часто відрізняються від характеристик навчального набору даних. Це зумовлено через відмінність у топології мережі, фоновому навантаженні та специфіці реалізації сценаріїв атак. Це явище, відоме як зсув домену або в англійській нотації (domain shift) [97]. Отже цей зсув призведе до неконтрольованого зростання помилок класифікації.

Для забезпечення стійкості дворівневої системи IDS, на наш розсуд слід математично описати відмінності між навчальним та реальним трафіком. Введемо математичний опис простору ознак та імовірнісних розподілів.

Нехай $\mathcal{X} \in \mathrm{R}^{\mathrm{d}}$, де d - вимірний простір ознак мережевих потоків, як-от тривалість потоку, кількість пакетів, ентропія заголовків тощо.

Позначимо через $\mathcal{D}_S$ домен навчання, який описує спільний розподілом ймовірностей $P_S(X, Y)$, де $X \in \mathcal{X}$, а Y – мітка класу.

Аналогічно, $\mathcal{D}_T$ – цільовий домен експлуатації з розподілом $P_T(X, Y)$.

У задачі виявлення вторгнень ми стикаємося з проблемою зміщення коваріат, коли умовні ймовірності класів збережені $P_S(Y|X) \approx P_T(Y|X)$, але маргінальні розподіли ознак суттєво відрізняються. Тобто $P_s(X) \neq P_T(X)$.

Для кількісної оцінки цього зміщення та обґрунтування необхідності налаштування порогів класифікації пропонуємо використовувати міру віддаленості розподілів на основі дивергенції Кульбака-Лейблера [39; 118].

Оскільки мережевий трафік, зазвичай, описують набором різнорідних ознак, як неперервних тау й дискретних, оцінку зсуву домену проводимо покомпонентно. Розглянемо j-ту ознаку $x^{(j)}$ як випадкову величину.

Нехай $P\left(x^{(j)}\right)$ – розподіл ймовірностей j-ї ознаки у навчальному наборі. Тобто еталон. А $Q\left(x^{(j)}\right)$ – розподіл тієї ж ознаки у поточному вікні

спостереження. Тобто емпіричні дані. Тоді дивергенцію Кульбака-Лейблера (або KL-дивергенцію) від Q до P визначимо як міра інформаційних втрат при апроксимації істинного розподілу P розподілом Q.

Для неперервних ознак, які попередньо дискретизовані, тобто розбиті на K інтервалів-бінів для синтезу гістограм, формула [118] набуде нового вигляду:

$$D_{KL}(P^{(j)} \| Q^{(j)}) = \sum_{k=1}^{K} P\left(x_k^{(j)}\right) \ln\left(\frac{P\left(x_k^{(j)}\right)}{Q\left(x_k^{(j)}\right)+\epsilon}\right) \quad (2.33)$$

Де $x_k^{(j)}$ – значення j-ї ознаки, що потрапляє у k-й інтервал дискретизації;

$P\left(x_k^{(j)}\right)$ – відносна частота (ймовірність) потрапляння ознаки в k-й інтервал у навчальній вибірці USB-IDS-1;

$Q\left(x_k^{(j)}\right)$ – відносна частота потрапляння ознаки в k-й інтервал у тестовій вибірці Це власний набір;

ϵ – параметр згладжування, $\epsilon \to 0^+$, що вводиться для уникнення невизначеності ділення на нуль у випадках, коли певні значення ознак атаки відсутні у тестовій вибірці. Тобто це абсолютна неперервність мір.

Значення $D_{KL} \approx 0$ свідчить про те, що профіль трафіку в мережі ідентичний навчальному. Відповідно класифікатор працюватиме з максимальною ефективністю. Зростання D_{KL} вказує на атипове відхилення у структурі даних. А своєю чергою це вимагатиме переходу системи у стан підвищеної невизначеності.

Оскільки IDS оперує багатовимірним вектором ознак, для прийняття рішення щодо достовірності класифікації атаки агрегуємо показники дивергенції. Ми пропонуємо застосувати зважену суму дивергенцій для формування вектора зсуву домену:

$$S_{domain} = \sum_{j=1}^{m} w_j \cdot D_{KL}(P^{(j)} || Q^{(j)}) \quad (2.34)$$

Де m – загальна кількість ознак атаки, що використовує класифікатор;

w_j – ваговий коефіцієнт j-ї ознаки, який визначаємо на основі її важливості, отриманої з моделі Random Forest наведеної вище в дисертації на етапі навчання.

Нормування ваг реалізуємо за умови $\sum_{j=1}^{m} w_j = 1$. Використання ваг w_j є необхідним, оскільки зсув розподілу малоінформативної ознаки, як-от випадкового порту джерела, не повинен вплинути на оцінку надійності системи так само сильно, як зсув критичної ознаки. Для прикладу, такою критичною ознакою атаки є довжина пакету або інтервал між пакетами.

Враховуючи виведену метрику $\mathcal{S}_{domain}$, модифікуємо вирішальне правило другого рівня системи Random Forest. Класична модель нагадаємо приймає рішення на основі максимізації апостеріорної ймовірності класу C_i. Тобто $\hat{y} = argmax_{C_i \in \mathcal{C}} P(C_i|X)$

У запропонованій в поточному параграфі дисертації моделі вводимо адаптивний поріг впевненості $\tau(\mathcal{S})$, який залежить від величини зсуву домену. Тоді правило класифікації набуває вигляду:

$$Decision(X) = \begin{cases} \hat{y}, \ max\ P(C_i|X) > \tau_{base} + \alpha \cdot \mathcal{S}_{domain}; \\ Unknown\ (Reject), \ \text{інакше}, \end{cases} \quad (2.35)$$

де τ_{base}– базовий поріг впевненості, визначений на валідаційній вибірці. Наприклад, 0.5;

α – коефіцієнт чутливості системи до зміни статистики трафіку, тобто гіперпараметр налаштування IDS;

$\mathcal{S}_{domain}$ – поточне значення інтегрального зсуву домену для часового вікна.

Тобто запропонований математичний апарат дозволяє розв’язати задачу перенесення навчених моделей у нові мережеві середовища. Далі обґрунтуємо налаштування порогів класифікації на основі мінімізації ризику в умовах зміщення розподілів.

У Баєсівській теорії прийняття рішень оптимальний поріг класифікації τ визначають співвідношенням вартостей помилок. Для бінарної класифікації, як от норма чи атака поріг τ_{bin} визначимо так:

$$\tau_{bin} = \frac{C_{FP}}{C_{FN}+C_{FP}} \quad (2.36)$$

Де C_{FP} – вартість помилкового спрацювання;

C_{FN} – вартість пропуску атаки.

Однак, вираз (2.55) базується на припущенні, що оцінка ймовірності $P(y|x)$, яку видає класифікатор, є каліброваною і точною. В умовах

зсуву домену. Тобто domain shift, коли розподіл вхідних даних $P_T(x)$ відрізняється від навчального $P_S(x)$, вихідна ймовірність моделі стає зміщеною. Використання статичного порогу τ_{bin} у такому середовищі призведе до неконтрольованого зростання ризику пропустити атаку.

Для розв'язання цієї проблеми пропонуємо модель порогу, яка скоригує точку прийняття рішення в залежності від величини інформаційної розбіжності між доменами.

Введемо поняття «ефективної вартості помилки» в цільовому середовищі. Припустимо, що зі зростанням відмінності поточного трафіку від навчального, тобто фіксуємо зростання метрики $\mathcal{S}_{domain}$, ризик невірної інтерпретації класифікатором зростає. Математично це опишемо через введення коригуючого коефіцієнта невизначеності $\lambda(\mathcal{S})$.

Нехай $\mathcal{S}_{domain}$ – зважена дивергенція Кульбака-Лейблера, розрахована для поточного часового вікна. Див. вирази (2.52) – (2.54).

Визначимо поріг для класу «Unknown» τ_{reject} як функцію від зсуву домену.

Базовий поріг відмови, який для ідеальних умов $\mathcal{S} \to 0$, виводимо з умови рівності ризиків прийняття хибного рішення та ризику відмови:

$$\tau_0 = 1 - \frac{C_{reject}}{C_{misclass}}, \tag{2.37}$$

де C_{reject} – вартість опрацювання події оператором «Unknown»,

$C_{misclass}$ – вартість помилки класифікації атаки.

В умовах зсуву домену пропонуємо моделювати зростання потенційної вартості помилки $C_{misclass}$ пропорційно до нечіткого середовища. Тобто:

$$\tilde{C}_{misclass}(\mathcal{S}) = C_{misclass}(1 + \beta \cdot \mathcal{S}_{domain}) \tag{2.38}$$

Де $\beta > 0$ – коефіцієнт чутливості моделі до змін середовища, який ми далі визначаємо в дисертації експериментально на етапі валідації моделі.

Підставивши модифіковану вартість у рівняння порогу, отримуємо закон налаштування порогу відмови:

$$\tau_{adaptive}(\mathcal{S}) = 1 - \frac{C_{reject}}{C_{misclass}} \cdot (1 + \beta \cdot \mathcal{S}_{domain}) \tag{2.39}$$

Розглянемо граничні умови запропонованої формули (2.39) для перевірки її адекватності:

Стаціонарне середовище $\mathcal{S}_{domain} \rightarrow 0$. Відповідно:

$$\tau_{adaptive} \rightarrow 1 - \frac{C_{reject}}{C_{misclass}} = \tau_0 \quad (2.40)$$

Порогове значення збігається з класичним Баєсівським порогом. Система працює в штатному режимі.

Зсув домену $\mathcal{S}_{domain} \rightarrow \infty$. Відповідно знаменник дробу зростає, отже, дріб прямує до 0, а сам поріг $\tau_{adaptive} \rightarrow 1$.

Це означає, що при суттєвій зміні структури трафіку поріг впевненості, необхідний для автоматичного рішення, наближається до 100%. Система перейде у режим «fail-safe», маркуючи більшість підозрілих подій як «unknown», замість того щоб робити ненадійні передбачення.

Для подальшої реалізації у рамках відповідної інформаційної технології та у програмному коді, див. розділ 3, розрахунок порогу виконуємо синхронно для кожного часового вікна Δt. Послідовність етапів наступна:

Етап 1. Розраховується вектор розподілів ознак $Q\_t(x)$ для поточного вікна.

Етап 2. Обчислюємо інтегральну метрику зсуву.

$$\mathcal{S}_t = \sum_{j=1}^{m} w_j \; D_{KL}\left(P^{(j)} \parallel Q_t^{(j)}\right) \quad (2.41)$$

Етап 3. Оновлюємо поріг прийняття рішень для другого рівня класифікатора – Random Forest: $\Theta_t = \max\left(\tau_{min}, \tau_{adaptive}(\mathcal{S}_t)\right)$

Етап 4. Проводимо класифікацію потоку x_i ^{x_i}, використовуючи правило:

$$\widehat{y_i} = \begin{cases} arg\ max_k\ P(C_k \mid x_i), & \text{якщо } max_k\ P(C_k \mid x_i) \geq \Theta_t; \\ Unknown, & \text{якщо } max_k\ P(C_k \mid x_i) < \Theta_t \end{cases} \quad (2.42)$$

Відповідно прийнято, що t_{bin} – поріг бінарної класифікації. Параметр $t_{adaptive}$ – синхронний поріг для багатокласової класифікації. Змінні $C_{FP}, C_{FN}, C_{reject}, C_{misclass}$ – елементи матриці вартості, які задаємо політикою безпеки. Та нарешті $\mathcal{S}_{domain}$ –скалярна величина зсуву домену на основі KL-дивергенції, див. вирази (2.41) – (2.42). А також β – параметр регуляризації, який контролює ступінь консервативності системи. Тобто чим вище β, тим частіше система відмовляється від рішення в нових умовах реалізації атаки.

Резюмуючи математичні моделі в параграфі 2.4 зазначимо наступне. Удосконалено метод виявлення вторгнень шляхом розробки математичної моделі налаштування порогів класифікації, яка, на відміну від чинних, базується на критерії мінімізації баєсівського ризику та враховує кількісну оцінку стохастичного зсуву домену на основі зваженої дивергенції Кульбака-Лейблера, що дозволяє забезпечити автоматичне коригування чутливості системи розпізнавання вторгнень до змін параметрів мережевого трафіку та мінімізувати вартість помилкових рішень в умовах варіативності атаки.

2.5 Інформаційна технологія реалізації та оцінювання дворівневої IDS

Формалізація інформаційних потоків

У межах запропонованої інформаційної технології реалізації дворівневої системи виявлення вторгнень базовою одиницею аналізу обрано мережевий потік (flow). Подальше оброблення та прийняття рішень здійснюється виключно на потоковому рівні, без використання пакетного представлення як об’єкта аналізу. Такий вибір зумовлений вимогами до масштабованості, інтерпретованості та формальної узгодженості з математичною моделлю класифікації, описаною в підрозділі 2.3.

Мережевий потік у межах інформаційної технології розглядається як агрегований опис мережевої взаємодії між кінцевими вузлами протягом обмеженого інтервалу часу. Кожен потік представляється у вигляді вектора числових ознак

$$f_i = (x_{i1}, x_{i2}, \dots, x_{im}) \in \mathbb{R}^m, \tag{2.43}$$

де компоненти вектора відображають статистичні, часові та об’ємні характеристики взаємодії, зокрема тривалість з’єднання, кількість пакетів, обсяг переданих даних, інтенсивність трафіку та співвідношення напрямків.

Саме такі вектори потокових ознак виступають елементарними об’єктами класифікації у дворівневій моделі прийняття рішень, формалізованій у підрозділі 2.3. Отже, інформаційна технологія безпосередньо оперує табличним набором даних, у якому кожен рядок відповідає одному об’єкту класифікації, а

кожен стовпчик – окремій ознаці. Це забезпечує пряму відповідність між типами даних інформаційної технології та припущеннями математичної моделі.

Вибір потокового представлення як базового формату даних обґрунтовується його перевагами над пакетним рівнем у контексті реалізації дворівневої IDS. Для формального підтвердження доцільності такого вибору в таблиці 2.1 наведено порівняння пакетного та потокового представлення за ключовими критеріями, що мають значення для побудови інформаційної технології та реалізації процедури класифікації.

Як видно з таблиці 2.1, саме потокове представлення забезпечує формальну сумісність між інформаційною технологією та математичною моделлю класифікації. Використання потоків дозволяє трактувати мережевий трафік як множину векторів ознак у багатовимірному просторі, що є необхідною умовою для реалізації дворівневої процедури прийняття рішень і подальшого оцінювання її ефективності [17].

Таким чином, у межах запропонованої інформаційної технології мережеві потоки обрано як базові інформаційні об’єкти аналізу, оскільки вони забезпечують компактне, інтерпретоване та формально узгоджене подання даних, придатне для застосування дворівневої моделі класифікації.

У межах запропонованої інформаційної технології оброблення мережевого трафіку організовано за принципом дискретної часової сегментації, відповідно до якої аналіз здійснюється над послідовністю фіксованих часових інтервалів. Така організація не орієнтована на безперервне відстеження часової еволюції мережі, а використовується як інструмент структуризації експериментальних сценаріїв, у межах яких система виявлення вторгнень функціонує в різних режимах навантаження та типах трафіку.

Таблиця 2.1

Порівняння пакетного та потокового представлення мережевого трафіку в контексті дворівневої IDS[1]

Критерій порівняння	Пакетне представлення	Потокове представлення
Рівень деталізації	Окремі пакети без контексту сесії	Агрегований опис мережевої взаємодії
Обсяг даних	Дуже великий, пропорційний кількості пакетів	Значно менший за рахунок агрегації
Обчислювальне навантаження	Високе, особливо для високошвидкісних мереж	Помірне, придатне для обробки в реальному часі
Формування поведінкових ознак	Обмежене, потребує складної попередньої обробки	Природне формування статистичних і часових ознак
Стійкість до шуму	Низька, окремі пакети легко спотворюють аналіз	Вища завдяки усередненню в межах потоку
Придатність до ML-аналізу	Обмежена без додаткової агрегації	Висока, відповідає табличному формату ознак
Інтерпретованість результатів	Складна для оператора IDS	Більш наочна та зрозуміла
Масштабованість	Обмежена на великих обсягах трафіку	Висока за рахунок зменшення кількості об’єктів
Відповідність дворівневій архітектурі	Низька – важко реалізувати фільтрацію	Висока – потоки природно передаються між рівнями
Придатність для віконної агрегації	Потребує додаткових перетворень	Безпосередньо підтримується

[1]*Джерело: сформовано автором*

Нехай Δt– фіксована тривалість часового вікна. Тоді часову вісь спостереження подано у вигляді множини неперекривних інтервалів

$$T = \{[t_0 + k\Delta t,\ t_0 + (k+1)\Delta t) \mid k \in \mathbb{N}\}. \quad (2.44)$$

Кожне часове вікно розглядається як елементарний фрагмент окремого сценарію мережевої взаємодії, в межах якого формується множина мережевих потоків

$$W_k = \{f_1^{(k)}, f_2^{(k)}, \dots, f_{n_k}^{(k)}\}, \quad (2.45)$$

де n_k– кількість потоків, зафіксованих у k-му часовому вікні відповідного сценарію.

Для кожного потоку $f_i^{(k)} \in W_k$ формується вектор ознак

$$f_i^{(k)} = (x_{i1}^{(k)}, x_{i2}^{(k)}, \dots, x_{im}^{(k)}) \in \mathbb{R}^m, \quad (2.46)$$

який надалі використовується як об’єкт класифікації у дворівневій моделі, описаній у підрозділі 2.1. Таким чином, інформаційна технологія оперує не окремими потоками в довільні моменти часу, а упорядкованими наборами потокових ознак, що відповідають фіксованим сегментам експериментального сценарію.

Запровадження часових вікон дозволяє формально відокремити:

- локальні рішення, що приймаються для окремих потоків у межах одного фрагмента сценарію;

- інтегральну поведінку системи, що проявляється у зміні розподілу рішень між різними сценаріями або між послідовними фазами одного сценарію.

Результат роботи дворівневої IDS у межах k-го фрагмента сценарію може бути поданий як множина вихідних рішень

$$Y_k = \{y_1^{(k)}, y_2^{(k)}, \dots, y_{n_k}^{(k)}\}, y_i^{(k)} \in \{\text{normative}, \text{attack}_1, \dots, \text{attack}_c, \text{unknown}\}.$$

За такої інтерпретації часові вікна слугують засобом порівняльного аналізу сценаріїв, а не інструментом безперервного моніторингу. Це дозволяє оцінювати стабільність та узгодженість рішень дворівневої IDS у різних умовах функціонування, зокрема шляхом зіставлення агрегованих показників ефективності між сценаріями. Таким чином, використання часових вікон у межах інформаційної технології створює формальну основу для аналізу поведінки дворівневої IDS у розрізі експериментальних сценаріїв, забезпечуючи відтворюваність результатів і коректне порівняння різних режимів роботи системи.

Структура інформаційної технології реалізації дворівневої системи виявлення вторгнень

Структура запропонованої інформаційної технології реалізації дворівневої системи виявлення вторгнень побудована у вигляді послідовного конвеєра оброблення даних (рис. 2.2.), у межах якого мережеві потоки проходять фіксовану послідовність етапів перетворення, аналізу та інтерпретації. Такий підхід забезпечує структурованість оброблення, відтворюваність експериментів і чітке розмежування функціональних ролей окремих компонентів системи.

Рис. 2.2 – Структура інформаційної технології реалізації дворівневої IDS

(*Джерело: сформовано автором за* [19])

На першому етапі виконується групування мережевих потоків у межах часових вікон, визначених у підрозділі вище. Результатом цього етапу є множина потоків, структурована відповідно до сценарію мережевої взаємодії, що створює вхідні дані для подальшого аналізу. Часові вікна на цьому етапі виконують роль одиниць сценарного аналізу, а не механізму безперервного моніторингу.

Другий етап pipeline передбачає попередню обробку ознак, метою якої є приведення потокових векторів ознак до уніфікованого формату, придатного для застосування моделей класифікації. На цьому етапі не приймаються рішення щодо типу трафіку; він виконує суто підготовчу функцію та забезпечує стабільність подальшого аналізу в межах усіх сценаріїв.

Третій етап відповідає бінарній класифікації потоків, яка виконує роль первинного фільтра в інформаційній технології. На цьому рівні кожен потік інтерпретується як такий, що належить до нормативного або потенційно небезпечного трафіку. У процесній моделі цей етап є точкою першого прийняття рішення та забезпечує скорочення обсягу даних, що передаються на наступний рівень аналізу.

Потоки, які не віднесено до нормативних, передаються на четвертий етап pipeline – багатокласову класифікацію з опцією відмови. Цей етап виконує функцію детальної інтерпретації типу атаки або формування стану невизначеності у випадках недостатньої впевненості моделі. Важливо, що в межах процесної моделі механізм відмови розглядається не як помилка класифікації, а як повноцінний результат оброблення, що має окреме семантичне значення.

П’ятий етап pipeline пов’язаний з оцінюванням результатів оброблення, яке інтегроване безпосередньо в інформаційну технологію. На цьому етапі формуються як класичні показники якості класифікації (accuracy, recall, F1), так і операційні метрики (coverage, reject rate), що дозволяють аналізувати поведінку системи в межах окремих сценаріїв.

Завершальним етапом процесної моделі є фіксація результатів оброблення, що передбачає їх збереження у структурованому вигляді та підготовку до подальшого аналізу. На цьому етапі формуються агреговані представлення результатів класифікації, які використовуються для порівняння експериментальних сценаріїв та узагальнення поведінки системи в межах інформаційної технології.

Узагальнюючи, процесна модель інформаційної технології визначає формальний порядок включення дворівневої моделі прийняття рішень у загальний цикл оброблення потокових даних. Послідовна організація pipeline задає фіксовану структуру взаємодії між етапами формалізації даних, класифікації та оцінювання, що дозволяє розглядати запропоновану систему як цілісну інформаційну технологію з визначеними входами, виходами та правилами перетворення інформації.

Дворівнева модель класифікації, формалізована в підрозділі 2.1, у межах інформаційної технології виконує не лише функцію прийняття рішень, але й визначає структуру оброблення даних та розподіл обчислювальних ролей. Кожен рівень моделі інтегровано в інформаційну технологію як окремий функціональний компонент із чітко визначеним призначенням, вхідними та вихідними даними.

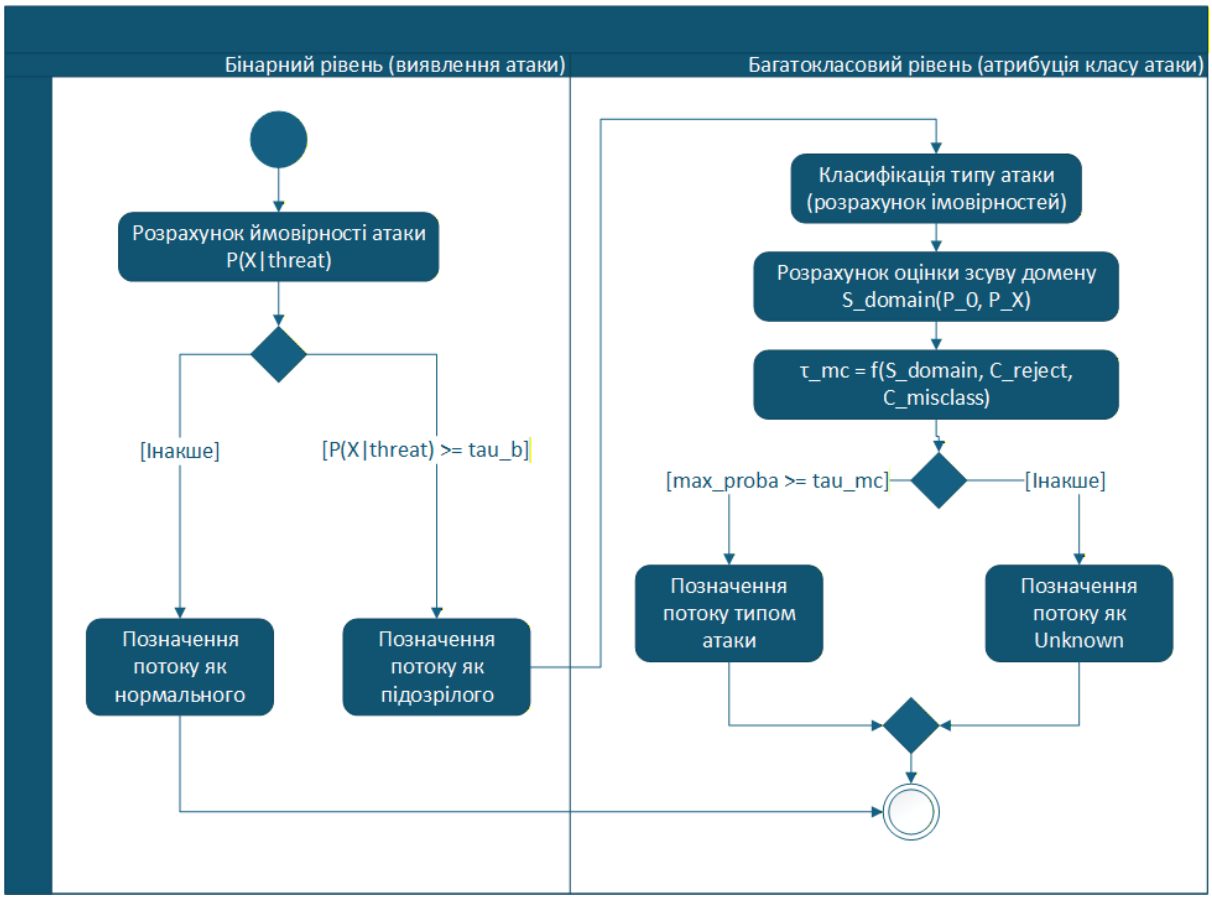

Рис. 2.3 – Діаграма дворівневої моделі прийняття рішень із адаптивним порогом на основі оцінки зсуву домену

(*Джерело: сформовано автором за* [19])

Функціональний розподіл ролей між рівнями дворівневої моделі в межах інформаційної технології подано на рис. 2.3.

Перший рівень дворівневої моделі виконує функцію первинної фільтрації мережевих потоків. У процесній моделі інформаційної технології цей рівень розглядається як механізм швидкого відокремлення потоків, що не потребують подальшого детального аналізу, від потоків, потенційно пов'язаних із аномальною або шкідливою поведінкою.

Функціонально перший рівень:

- зменшує обсяг даних, що передаються на наступні етапи оброблення;
- забезпечує раннє відсікання нормативного трафіку;
- стабілізує роботу pipeline за рахунок обмеження навантаження на другий рівень.

У межах інформаційної технології перший рівень не інтерпретує тип атаки та не формує остаточних висновків щодо характеру загрози. Його роль обмежується прийняттям рішення про доцільність подальшої обробки потоку, що відповідає концепції фільтра в багатоступеневих системах аналізу.

Другий рівень дворівневої моделі інтегровано в інформаційну технологію як механізм семантичної інтерпретації потоків, відібраних на попередньому етапі. На відміну від першого рівня, цей компонент орієнтований не на скорочення обсягу даних, а на деталізацію результатів аналізу.

Функціонально другий рівень:

- здійснює атрибуцію потоків до множини відомих класів атак;
- формує інтерпретовані результати, придатні для подальшого оцінювання;
- працює з обмеженою, попередньо відібраною підмножиною потоків.

У контексті інформаційної технології другий рівень не розглядається як універсальний класифікатор для всього трафіку. Його застосування обмежене лише тими потоками, які пройшли первинну фільтрацію, що дозволяє зосередити обчислювальні ресурси на складних і потенційно небезпечних випадках.

Механізм відмови від рішення (*reject / unknown*) є невід'ємним елементом запропонованої інформаційної технології та інтегрований безпосередньо в роботу другого рівня pipeline. У процесній моделі він розглядається не як побічний ефект недостатньої точності класифікації, а як повноцінний функціональний результат оброблення.

Функціональне призначення механізму відмови полягає в:

- явному відокремленні випадків недостатньої впевненості від помилкової атрибуції;
- зниженні ризику некоректної інтерпретації типу атаки;
- підтримці керованого компромісу між повнотою класифікації та надійністю результатів.

У межах інформаційної технології стани *unknown* акумулюються та враховуються на етапі оцінювання, що дозволяє аналізувати частку відмов у різних експериментальних сценаріях. Таким чином, механізм відмови виступає

інструментом контролю поведінки системи, а не лише допоміжним елементом класифікатора.

Інтеграція оцінювання в цикл оброблення

Оцінювання ефективності дворівневої системи виявлення вторгнень у межах запропонованої інформаційної технології інтегроване безпосередньо в цикл оброблення мережевих потоків і розглядається як функціональний компонент технології, а не як зовнішня процедура після завершення класифікації. Такий підхід дозволяє аналізувати не лише точність окремих рішень, але й поведінку системи в цілому в межах експериментальних сценаріїв.

У контрольованих умовах, за наявності еталонних міток класів, для оцінювання якості класифікації застосовуються класичні метрики, зокрема accuracy, precision, recall та F1-міра. У межах інформаційної технології ці показники використовуються як базові індикатори коректності прийняття рішень на окремих рівнях дворівневої моделі.

Важливо, що класичні метрики можуть обчислюватися як у повністю офлайн-режимі (на заздалегідь сформованих вибірках), так і в умовах покрокового оброблення потоків у межах часових вікон. У другому випадку вони не трактуються як характеристики стабільної якості моделі, а використовуються для локального аналізу результатів у межах конкретного сценарію.

Висновки до другого розділу

У другому розділі сформовано теоретичні та технологічні засади побудови дворівневої системи прийняття рішень у системі виявлення вторгнень із явним використанням порогових правил, що визначають умови автоматичної класифікації та відмови від рішення. На відміну від підходів, у яких пороги задаються евристично або підбираються емпірично, у роботі прийняття рішень у IDS розглянуто в межах теорії мінімізації очікуваного ризику з урахуванням асиметрії втрат.

У межах теоретичної частини розділу обґрунтовано дворівневу модель прийняття рішень, у якій кожен рівень має власний тип порогу та відповідає за контроль різних видів ризику. Перший рівень реалізує порогове правило первинної детекції, оптимізоване з урахуванням високої вартості пропуску атак. Аналітично показано, що оптимальне рішення на цьому етапі визначається

порівнянням апостеріорної ймовірності атаки з порогом, який задається співвідношенням вартостей помилок типу false negative та false positive. Таким чином, поріг першого рівня формалізує допустимий компроміс між чутливістю системи та кількістю хибних спрацьовувань і виступає параметром політики IDS, а не властивістю конкретного класифікатора.

Для другого рівня дворівневої моделі аналітично обґрунтовано порогове правило атрибуції типу атаки з можливістю відмови від рішення. Уведення окремої вартості відмови дозволило сформулювати умову, за якої автоматична атрибуція є доцільною лише тоді, коли очікуваний ризик хибної класифікації не перевищує ризик відмови. За такої постановки клас невизначеності розглядається як повноцінний результат прийняття рішення, а не як помилка класифікації. Порогове правило другого рівня забезпечує керований компроміс між повнотою атрибуції та надійністю інтерпретації результатів у ситуаціях обмеженої впевненості моделі.

Показано, що пороги першого та другого рівнів утворюють узгоджену політику прийняття рішень у дворівневій IDS. Порогове правило першого рівня визначає обсяг трафіку, який передається на подальший аналіз, тоді як поріг другого рівня контролює доцільність автоматичної інтерпретації типу атаки. Такий підхід унеможливлює незалежне або ізольоване налаштування порогів і дозволяє інтерпретувати поведінку системи через параметри ризику, що мають чіткий операційний зміст.

У технологічній частині розділу показано, яким чином аналітично виведені порогові правила інтегруються в інформаційну технологію реалізації та оцінювання дворівневої IDS. Обґрунтовано вибір потокового представлення мережевого трафіку та дискретної часової сегментації як засобів, що дозволяють застосовувати порогові правила на практиці та забезпечують відтворюваність експериментальних сценаріїв.

Таким чином, другий розділ формалізує дворівневу IDS як систему з керованими порогами прийняття рішень, у якій автоматична класифікація та відмова від рішення визначаються аналітично на основі мінімізації очікуваного ризику. Отримані положення створюють методологічну основу для експериментального дослідження впливу порогових параметрів на поведінку системи та оцінювання її ефективності в різних сценаріях мережевого трафіку, що розглядається у наступному розділі.

РОЗДІЛ 3
РОЗРОБКА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ТА ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА ЕФЕКТИВНОСТІ ВИЯВЛЕННЯ МЕРЕЖЕВИХ АТАК

У даному розділі представлено практичну реалізацію інтелектуальної системи виявлення атак у віртуальному середовищі, орієнтовану на автоматичну обробку мережевого трафіку та багаторівневу класифікацію аномальних потоків. На відміну від теоретичних або статистичних підходів, акцент зроблено на застосуванні повнофункціонального прототипу, здатного працювати у реальному або симульованому часовому масштабі з подальшим аналізом результатів.

Побудова системи охоплює кілька ключових напрямів. По-перше, здійснено розробку тренувальних модулів (тренерів) для моделей машинного навчання, які забезпечують коректне навчання алгоритмів на основі заздалегідь оброблених та уніфікованих наборів даних. По-друге, сформовано повністю ізольоване віртуальне середовище, в якому створено умови для запуску як нормативного, так і зловмисного трафіку без ризику для зовнішньої інфраструктури. По-третє, реалізовано програмний засіб збору та класифікації потоків трафіку, який інтегрує двоступеневу модель виявлення: первинну бінарну фільтрацію та подальшу ідентифікацію типу загрози. Нарешті, розроблено і протестовано процедуру запуску різних сценаріїв DDoS-атак, з наступним збереженням та інтерпретацією результатів у вигляді структурованих вихідних файлів.

Реалізація системи у замкненому лабораторному середовищі дає змогу перевірити її працездатність, відтворюваність результатів та адаптованість до змішаного трафіку, що містить як легітимні, так і шкідливі компоненти.

3.1. Побудова тренувальних модулів Naive Bayes та Random Forest

Тренувальний модуль починає обробку даних із завантаження навчального набору, за необхідності поетапно для великих файлів (див. рис. 3.1). Використовується підхід читання CSV-файлів частинами за допомогою програмної бібліотеки pandas, що дозволяє обробляти навіть дуже великі набори

даних, тримаючи в пам’яті лише порцію записів за раз [65]. Така потокова обробка запобігає переповненню пам’яті та пришвидшує роботу з великими файлами.

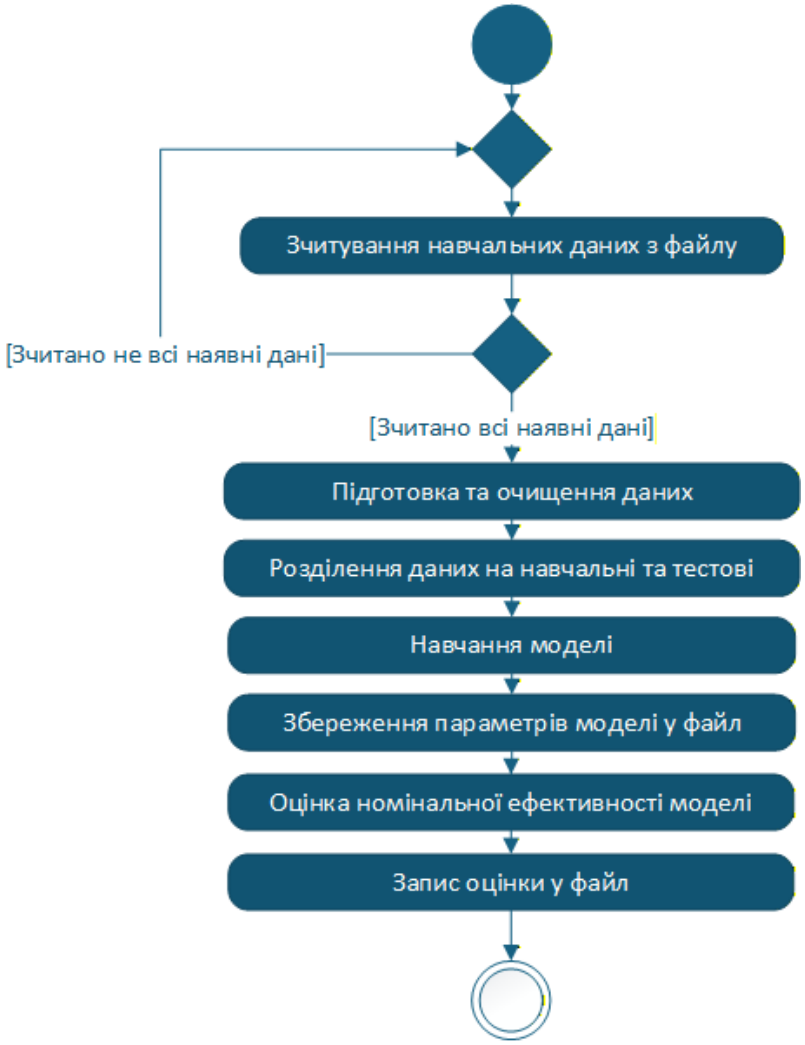

Рис. 3.1. Діаграма діяльності алгоритму навчання моделі ML

(*Джерело: сформовано автором за* [81])

Етап підготовки даних у конвеєрі машинного навчання зазвичай включає очищення даних, обробку пропусків і відбір істотних характеристик [69]. Застосування цих практик забезпечує, що модель навчатиметься лише на релевантних та якісних даних (див. підрозділ 2.5). На цьому етапі дані також відкидаються непотрібні стовпці, такі як службові ідентифікатори на кшталт IP-адрес чи портів. Ці дані не несуть корисного навантаження з точки зору класифікації – різні мережі можуть мати різну конфігурацію, а отже пакети і потоки пакетів недоцільно класифікувати за такими параметрами. Також у випадку обробки власноруч зібраних даних видаляються прогнозні поля

(наприклад, поля predicted, probability_safe, probability_threat з попередніх класифікацій). Раннє вилучення нерелевантних ознак зменшує обсяг даних і пам’яті, необхідної для наступних кроків, що є важливою оптимізацією при роботі з великими вибірками (такими як USB-IDS-1).

У випадку числових ознак може здійснюватися нормалізація або масштабування, щоб привести різні показники до співставного масштабу. Хоча в реалізованому модулі явне масштабування не виконувалося, була застосована інша техніка зменшення розмірності – відсів ознак із низькою варіативністю. Зокрема, після очистки даних була обчислена дисперсія кожної ознаки, і всі ознаки з варіацією менше заданого порогу (в розглянутому алгоритмі 0.01) були видалені. Такий підхід еквівалентний використанню фільтру VarianceThreshold, який вилучає малоінформативні ознаки з майже нульовою варіацією [66]. Вилучення маловаріативних параметрів скорочує розмірність задачі і може покращити продуктивність як в часі навчання, так і в точності, усуваючи «шумові» ознаки.

Перед розділенням даних на навчальні та тестові було здійснено кодування міток у формат, придатний для навчання моделі. Використовується клас LabelEncoder з бібліотеки Scikit-learn для перетворення текстових міток у цілочисельні значення [41]. Така трансформація є стандартною практикою, оскільки багато алгоритмів машинного навчання працюють лише з числовими мітками класів. Зокрема, для задачі мережевої безпеки всі значення мітки Label (наприклад, BENIGN, Slowloris, Hulk тощо) кодуються як $0, 1, 2, \ldots$ для відповідних класів. У випадку двокласової моделі (як у Naive Bayes тренері) мітки спрощуються до двох категорій: «нормальна» та «атака» – перед їх числовим кодуванням. В результаті отримуємо вектор цільових значень $y = \{0, 1\}$, де 0 – нормальний трафік, 1 – атака.

Розбиття на навчальну та тестову вибірки: Після підготовки ознак і кодування міток, дані поділяються на навчальний та тестовий піднабори. Як рекомендується у стандартах побудови ML-процесів, виділення тестової вибірки є критично важливим для об’єктивної оцінки моделі [8]. У реалізації модулів було використано класичне співвідношення: 80% даних – для навчання, 20% –

для тестування. Розбиття здійснюється випадковим чином з фіксованим $random\_state = 42$ для відтворюваності результатів. За рахунок перемішування (shuffle) перед розділенням забезпечується статистична однорідність вибірок. Такий підхід узгоджується з найкращими практиками, де тренувальні дані використовуються для побудови моделі, а тестові – лише для фінальної перевірки узагальнювальної здатності класифікатора[8].

Навчання моделі з урахуванням особливостей алгоритму: Після отримання підготовлених наборів X_train, y_train (для навчання) та X_test, y_test (для перевірки) розпочинається власне тренування моделі. Алгоритм Gaussian Naive Bayes та Random Forest мають дещо різні вимоги та можливості, тому тренувальні модулі враховують ці відмінності:

Модель GaussianNB навчається або на всіх даних одразу, або – у випадку дуже великого обсягу даних – пакетно. Застосовано інкрементальне навчання методом partial_fit для випадків, коли розмір вибірки перевищує встановлений поріг (наприклад, >50 тисяч зразків). Інкрементальне навчання означає, що модель оновлює параметри поступово, читаючи дані частинами, що дозволяє навчатися на датасетах, які не поміщаються повністю в оперативну пам’ять [13].

В свою чергу модель RandomForestClassifier зазвичай не підтримує простого інкрементального навчання, тому реалізовано іншу стратегію. Спочатку проводиться тренування базового лісу з заданими гіперпараметрами, після чого – пошук оптимальних гіперпараметрів методом випадкового пошуку. Використано RandomizedSearchCV з кількістю ітерацій n_iter=5 і крос-валідацією на 3 етапи для вибору найкращої комбінації параметрів за метрикою *повноти* (recall). Вибір RandomizedSearchCV замість традиційного GridSearchCV обумовлений ефективністю: випадковий пошук тестує лише випадкову підмножина всіх комбінацій, що суттєво скорочує обчислювальні витрати на великому просторі параметрів [64]. Згідно з документацією Scikit-learn, RandomizedSearchCV досліджує задану кількість випадкових комбінацій параметрів із заданих діапазонів, що часто майже не поступається повному перебору за якістю, але значно швидше знаходить близький до оптимуму варіант [9]. У тренувальному модулі Random Forest параметри для пошуку включали, зокрема, число дерев

(n_estimators), максимальну глибину дерева (max_depth), мінімальний розмір вузла поділу (min_samples_split) та максимальну частку ознак при розщепленні (max_features). Обмеження діапазонів цих параметрів до кількох адекватних значень також було зроблено з метою зниження навантаження на пам’ять та процесор. Після виконання RandomizedSearchCV модель налаштовується на найкращу знайдену комбінацію параметрів і донавчається на всьому X_train. Таким чином, тренувальний модуль Random Forest поєднує стандартне навчання з моделлю із пошуком оптимальних гіперпараметрів, що підвищує її узагальнюючу здатність. Гіперпараметри моделі для рідних наборів даних наведено у таблиці 3.1.

Таблиця 3.1

Гіперпараметри моделі Random Forest після навчання[2]

Показник	RF/USB-IDS-1	RF/Власні дані
Обсяг даних	999 621 запис	41 850 записів
Max depth	10	30
Max features	sqrt	log2
Min samples split	100	100
n_estimators	100	100
Час навчання моделі	0:07:18.831685	0:00:05.224560

[2]*Джерело: сформовано автором*

Перед серіалізацією тренувальний модуль проводить оцінювання моделі на відкладеній тестовій вибірці. Обчислюються основні метрики класифікації – точність (accuracy), точність позитивного прогнозу (precision), повнота (recall) та F1-міра. Метрики виводяться в консоль та можуть бути записані для аналізу ефективності моделі. Важливо підкреслити, що для багатокласової моделі (Random Forest) критичною метрикою було обрано повноту (macro-Recall) – саме вона використовувалася як цільова при підборі гіперпараметрів. Такий акцент виправданий доменною областю (виявлення аномалій та атак): у кібербезпеці пропуск навіть однієї загрози може мати фатальні наслідки, тому модель має виявляти максимум шкідливих дій навіть ціною хибних спрацьовувань [47]. Іншими словами, максимізація повноти дозволяє зменшити кількість

невиявлених атак (мінімізувати False Negative), що є пріоритетом для систем виявлення вторгнень [47].

Завершальним кроком тренування є збереження моделі і пов'язаних з нею об'єктів на диск. Після навчання модель серіалізується за допомогою модуля pickle у файл, щоб її можна було завантажити для подальшого використання без повторного навчання. Разом з моделлю зберігається і об'єкт LabelEncoder (який містить відповідність між оригінальними категоріями міток і числовими кодами). Збереження енкодера гарантує, що при використанні моделі на нових даних (в режимі класифікації) можна буде коректно перетворити вхідні мітки класів у потрібний формат. Такий підхід відповідає вимозі відтворюваності, щоб усі параметри моделі могли бути відтворені і використані під час класифікації даних, тому їхні налаштування зберігаються разом з моделлю [69]. Характеристики та номінальні оцінки ефективності моделей наведено в таблиці 3.2.

Таблиця 3.2.

Характеристика та розрахункові оцінки моделей Naïve Bayes та Random Forest для наборів даних USB-IDS-1 та власного набору даних[3]

Набір даних	**USB-IDS-1**		**Власні дані**	
Тип алгоритму	Naïve Bayes	Random Forest	Naive Bayes	Random Forest
Розмір набору даних	4 813 420		41 850	
Кількість оброблених записів	800 000		33 480	
Кількість ознак	70	70	65	65
Accuracy	0.98127	0.99995	0.91816	1.00000
Precision	0.89856	0.99886	0.85626	1.00000
Recall	0.95982	0.99961	0.94714	1.00000
F1-score	0.92660	0.99923	0.88872	1.00000
Час підготовки даних	≈ 1 хв 53 с	≈ 0.91 с	≈ 0.03 с	≈ 0.03 с
Час навчання моделі	<1сек	7 хв 19 с	<1сек	5 с

[3]*Джерело: сформовано автором*

Також на рисунках 3.2 і 3.3 зображено матриці помилок для моделей, навчених на різних наборах даних. Відповідно до значень в цих матрицях можемо зробити висновок, що моделі для обох рівнів класифікують дані з тестової

підвибірки з високою точністю (виявлення атаки на першому рівні близьке до 1, атрибуція класу атаки на другому рівні дорівнює 1 для всієї тестової підвибірки).

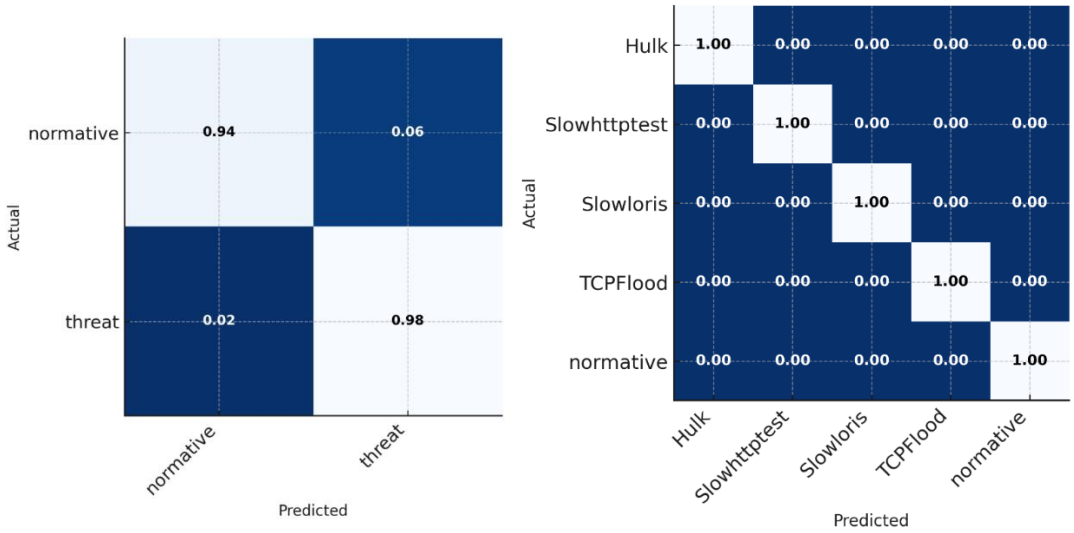

Рис. 3.2. Відносні матриця невідповідностей для моделі навченої на наборі USB-IDS-1 (зліва – для бінарної класифікації, справа – для багатокласової) (*Джерело:сформовано автором)*

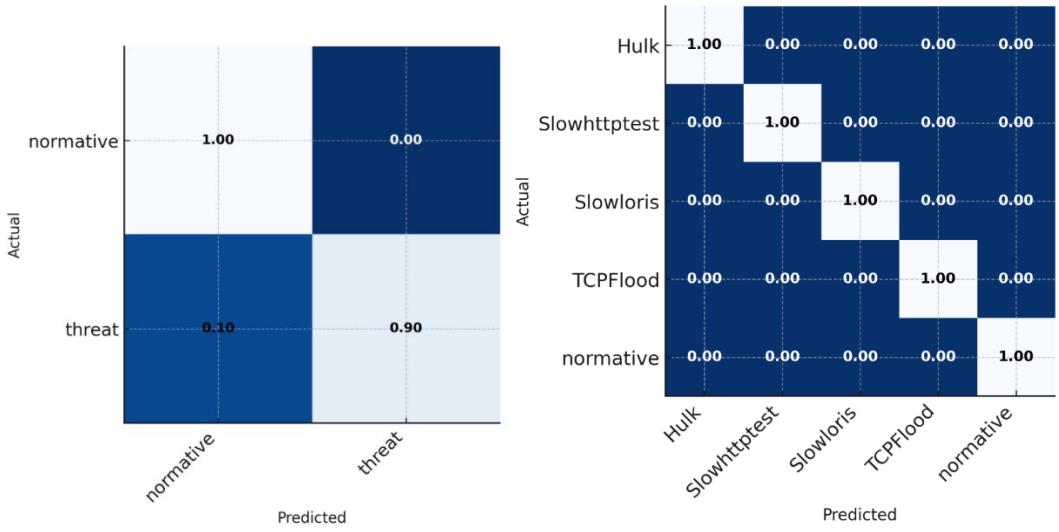

Рис. 3.3. Відносні матриця невідповідностей для моделі навченою на власному наборі даних (зліва – для бінарної класифікації, справа – для багатокласової) (*Джерело:сформовано автором)*

Отримані значення не слід інтерпретувати як універсальні; вони відображають ефективність моделі в контрольованому експериментальному середовищі та використовуються для порівняння архітектурних рішень.

Реалізація тренувальних модулів спроєктована з дотриманням принципів модульності та повторного використання коду. Базовий клас Trainer визначає узагальнений інтерфейс та ключові методи, спільні для будь-якої моделі, а спеціалізовані класи-нащадки (TrainerBayes, TrainerRandomForest) розширюють його, перевизначаючи лише необхідну логіку. Такий об’єктно-орієнтований дизайн забезпечує низку важливих переваг:

- Чіткий інтерфейс: клас Trainer задає стандартні методи, зокрема: init_data() для завантаження даних, prepare_data() для очищення та підготовки датасету, prepare_data_for_training() для формування train/test вибірок і кодування міток, test_model() для оцінки моделі на тестових даних. Наявність єдиного набору методів означає, що всі тренери дотримуються однакової структури роботи. Це полегшує їх використання у вищих модулях системи: наприклад, можна викликати послідовно train_and_store() для будь-якого тренера, незалежно чи це Naïve Bayes чи Random Forest, і очікувати, що він виконає повний цикл навчання збереження моделі. Інтерфейс уніфіковано, а деталі реалізації приховані всередині конкретних класів.

- Наслідування реалізації: спеціалізовані тренери успадковують від базового не тільки методи, а й поля (наприклад, атрибут self.data, логіку збереження label_encoder_file тощо). Вони повторно використовують значну частину коду базового класу – зокрема, RandomForest-тренер викликає super().__init__(), що автоматично ініціалізує дані стандартним способом (завантажує всі CSV і проводить базове очищення). Таким чином, уникнуто дублювання коду: загальні кроки (як-от читання файлів та інтегральна підготовка) написані один раз у Trainer і застосовуються в обох модулях. Наприклад, методи роботи з мітками (використання LabelEncoder) реалізовані у базовому класі, тому і Naive Bayes, і Random Forest тренери користуються одним і тим самим механізмом кодування цільової змінної, гарантуючи узгодженість між ними. Такий підхід відповідає принципу «Write Once, Use Many» – написати код один раз і застосовувати у різних місцях, що підвищує надійність та однаковість роботи модулів [69].

- Перевизначення лише необхідного: кожен спеціалізований тренер змінює поведінку базового рівно там, де це потрібно його алгоритму. Зокрема, TrainerBayes не викликає super().__init__(), оскільки йому не підходить стандартне повне завантаження даних – замість цього він реалізує власний init_data() з покроковим читанням і обробкою блоків. Водночас TrainerRandomForest навпаки успадковує базове завантаження, але перевизначає process_files() для фільтрації benign-трафіку і обмеження вибірки. Обидва тренери імплементують свій train_and_store(): у баєсівському – з логікою пакетного навчання GaussianNB [18], у RandomForest– з запуском RandomizedSearchCV перед навчанням фінальної моделі. Таким чином, через механізм поліморфізму кожен підклас реалізує потрібні кроки по-своєму, не порушуючи загальної структури. Це робить систему гнучкою: додавання нової моделі (наприклад, тренер для SVM або нейронної мережі) вимагатиме лише створити новий клас, успадкований від Trainer, та переозначити специфічні етапи (можливо, preprocess, train або evaluate), зберігаючи інші методи базовими.

- Масштабованість та підтримуваність: Завдяки модульності, система легко розширюється – можна додавати нові алгоритми чи змінювати обсяг даних, не переписуючи з нуля весь конвеєр. Наприклад, якщо виникне потреба опрацювати інший датасет чи додати ще очистки, достатньо відкоригувати базовий метод prepare_data або process_data, і зміни автоматично застосуються до всіх тренерів, що його наслідують. Це значно спрощує супровід коду: виправлення помилки або оптимізація в одному місці (базовому класі) відразу впливає на всі модулі, що використовують цю функціональність. Така ізольованість компонентів спрощує налагодження – кожен модуль (завантаження, конвертація, навчання) можна тестувати окремо. Літературні джерела наголошують, що перехід від монолітних скриптів до модульної архітектури підвищує повторне використання коду, полегшує масштабування та спрощує командну роботу над проектом [50]. У нашому випадку це реалізовано через об’єктно-орієнтовану ієрархію тренерів: базовий функціонал багаторазово застосовується у підкласах, а їх розробка і розширення не потребує зміни вже перевіреного коду. У результаті забезпечено баланс між уніфікацією (єдина схема підготовки даних і оцінки) та

спеціалізацією (алгоритмо-орієнтовані оптимізації), що відповідає сучасним підходам до побудови структурованих ML-пайплайнів [50].

Таким чином, поряд з функціональністю код має інкапсульовану логіку, та є поліморфним, застосовуючи механізм наслідування класів.

Отже, продумана модульність тренувальних модулів Naive Bayes та Random Forest сприяє їхній надійності та ефективності. Повторне використання коду мінімізує ймовірність помилок і розбіжностей між реалізаціями, а чітке розділення відповідальності (завантаження даних, попередня обробка, навчання, оцінка) дозволяє легко модифікувати або доповнювати кожен компонент без побоювань порушити роботу інших. Ці принципи особливо важливі, адже забезпечують відтворюваність експериментів та легкість порівняння різних методів на однаковій основі.

3.2. Методика створення маркованих наборів даних мережевого трафіку

Однією з ключових проблем у дослідженнях інтелектуальних систем виявлення вторгнень є формування коректного ground truth для навчання та оцінювання моделей машинного навчання [104]. У більшості відомих наборів даних маркування мережевого трафіку здійснюється постфактум на основі часових міток експериментів, ролей вузлів або евристичних правил, що ґрунтуються на IP-адресах і портах. Такий підхід не гарантує однозначної відповідності між окремими мережевими потоками та активними сценаріями атак, особливо в умовах перекриття нормативного та атакувального трафіку або за наявності кількох одночасних атак [104].

Додатковою складністю є те, що значна частина досліджень оперує пакетним представленням даних або агрегованими статистиками без явної формалізації переходу від пакетів до потоків і відповідного перенесення міток [59]. У результаті маркування на рівні потоків часто є непрямим і залежить від припущень, що знижує відтворюваність і внутрішню узгодженість експериментів.

У зв’язку з цим актуальним є розроблення методики формування маркованого мережевого трафіку, в якій ground truth задається детерміновано та не потребує ручної або евристичної розмітки після завершення експерименту.

Загальна концепція запропонованої методики.

У даній роботі запропоновано методику формування маркованого мережевого трафіку, засновану на детермінованому маркуванні пакетів під час етапу їх генерації з подальшою автоматизованою агрегацією пакетних міток у потокове представлення.

Загальну структуру запропонованої методики формування маркованого мережевого трафіку на основі детермінованого пакетного маркування подано на рис. 3.4.

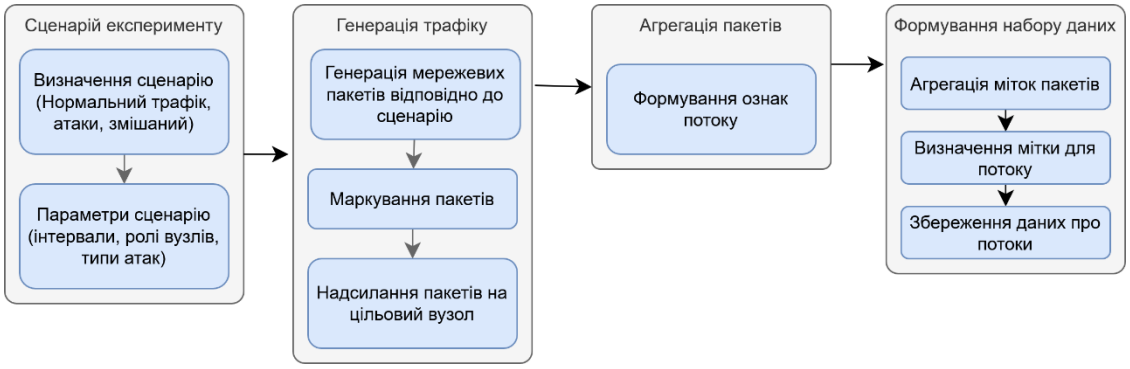

Рис. 3.4 – Структурна схема методики формування маркованого мережевого трафіку на основі детермінованого пакетного маркування та агрегації у потокове представлення (*Джерело:сформовано автором за* [83; 98])

Принциповою відмінністю методики є те, що мітка трафіку формується в момент створення мережевих пакетів, відповідно до активного сценарію, а не відновлюється після збору даних. За рахунок цього забезпечується внутрішньо узгоджений ground truth, який випливає з конструкції експерименту та контрольованого середовища.

Методика орієнтована на використання потокових ознак мережевого трафіку та призначена для досліджень систем виявлення вторгнень у віртуальних середовищах із можливістю моделювання змішаних атак.

Формальна модель даних та позначення.

Експериментальний процес організовується у вигляді часової осі, поділеної на послідовність неперекривних вікон фіксованої тривалості.

У межах кожного вікна формується множина мережевих потоків:

$$F_k = \{f_{k,1}, f_{k,2}, \dots, f_{k,n_k}\}. \quad (3.1)$$

Кожен потік f складається з множини мережевих пакетів:

$$P(f) = \{p_1, p_2, \dots, p_{m_f}\}. \quad (3.2)$$

Для кожного пакета p на етапі генерації визначається мітка класу:

$$y(p) \in \mathcal{Y} = \{\text{normative}\} \cup \{attack_1, \dots, attack_m\} \quad (3.3)$$

Після агрегації пакетів у потоки кожному потоку відповідає вектор статистичних ознак $x(f) \in \mathbb{R}^d$ та мітка потоку $y(f)$, що визначається за правилами, описаними нижче.

Ізольоване віртуальне середовище генерації трафіку

Генерація трафіку повинна здійснюватись в ізольованому віртуальному середовищі, яке унеможливлює появу стороннього мережевого трафіку та забезпечує повний контроль над ролями вузлів і сценаріями взаємодії. У середовищі визначено вузли з фіксованими ролями [1]:

- джерела нормативного трафіку;
- джерела атакувального трафіку;
- вузол збору та моніторингу трафіку.

Конфігурація середовища (топологія, маршрутизація, параметри каналів) задається до початку експерименту та зберігається для забезпечення відтворюваності результатів.

Детерміноване маркування пакетів на етапі генерації

У межах запропонованої методики кожен пакет, що генерується у віртуальному середовищі, маркується відповідно до активного сценарію мережевої взаємодії. Маркування здійснюється автоматично, без ручного втручання, та є детермінованим. Мітка може розміщуватись як в заголовках мережевих пакетів, так і в payload. Оскільки вузол мережі, який приймає трафік є контрольованим, він повинен мати чіткий набір правил по вилученню міток з мережевих пакетів.

Для нормативного трафіку всім пакетам присвоюється мітка normative. Для атакувального трафіку мітка пакета відповідає типу атаки, що генерується у

відповідний момент часу. Таким чином, кожен пакет несе інформацію про належність до певного сценарію, що забезпечує первинний ground truth на пакетному рівні.

Такий підхід дозволяє уникнути неоднозначностей, пов'язаних з подальшим відновленням міток за часовими або мережевими атрибутами, та забезпечує чіткий зв'язок між процесом генерації трафіку і маркуванням даних.

Агрегація пакетних міток у потокове представлення

Після збору трафіку пакети агрегуються у мережеві потоки на основі стандартних ідентифікаторів з'єднання та часових вікон. Для кожного потоку формується множина пакетних міток $Y_p(f) = \{y(p) \mid p \in P(f)\}$.

Мітка потоку визначається як детермінована функція множини $Y_p(f)$. У базовому варіанті методики використовується правило:

$$y(f) = \begin{cases} \text{attack}_j, & \text{якщо } \exists p \in P(f): y(p) = \text{attack}_j, \\ normative, & \text{якщо } \forall p \in P(f): y(p) = normative. \end{cases} \quad (3.4)$$

У разі наявності в потоці пакетів з різними атакувальними мітками застосовується наперед визначена політика інтерпретації, а саме вибір домінуючої мітки.

Таким чином, мітка потоку є прямим наслідком міток пакетів і не залежить від зовнішніх евристик або ручних правил.

Обробка змішаних сценаріїв атак

Методика допускає одночасну активність кількох атакувальних процесів, що призводить до формування змішаних сценаріїв мережевого трафіку. У таких умовах окремі мережеві потоки можуть містити пакети, згенеровані різними сценаріями.

Для коректної інтерпретації таких потоків у методиці передбачено використання явної політики маркування, яка визначається до початку експерименту. Це дозволяє уникнути неоднозначностей при подальшому навчанні та оцінюванні моделей IDS і забезпечує узгодженість результатів між різними експериментами.

Відтворюваність та обмеження методики

Запропонована методика забезпечує відтворюваність результатів за рахунок фіксації конфігурації віртуального середовища, параметрів сценаріїв та правил маркування. Ground truth формується детерміновано на основі пакетних міток і не потребує ручної розмітки.

Водночас методика не передбачає явної перевірки ефективності атак або оцінювання впливу окремих потоків на стан цільової системи. Маркування трафіку відображає належність до сценарію, а не фактичний рівень успішності атаки, що слід враховувати при інтерпретації результатів експериментів. Також варто зазначити, що додавання міток то згенерованого трафіку може незначно спотворювати деякі статистичні ознаки потоків (наприклад packet length).

3.3. Реалізація алгоритму збору та класифікації трафіку

В основі алгоритму – клас FlowCapture, що організовує збір мережевих потоків у режимі реального часу та їх періодичну пакетну обробку. Загальна послідовність така: з початком захоплення ініціюється безперервний цикл, який кожні window_size (60 за замовчування) секунд формує черговий набір потоків (вікно) для аналізу. Після закінчення інтервалу зібрані мережеві потоки цього вікна зберігаються у файл та піддаються обробці і класифікації. За результатами класифікації для вікна обчислюються показники (метрики) і дані записуються у відповідні вихідні файли. Далі цикл переходить до збору наступного вікна (поки не надійде сигнал завершення). Нижче детально розглянуто внутрішню логіку кожного етапу алгоритму.

При створенні об’єкта класу FlowCapture виконується початкова ініціалізація параметрів. Конструктор зберігає налаштування: мережевий інтерфейс для прослуховування трафіку (interface), тривалість часовго вікна в секундах (window_size) та директорію для вихідних файлів (output_dir). У процесі ініціалізації створюються необхідні структури директорій: зокрема, каталог для тимчасових файлів (наприклад, temporary_files) і каталог для остаточних результатів, де будуть зберігатись оброблені дані потоків.

Діаграма класів, які були розроблені для реалізації дворівневої моделі зображена на рис. 3.5.

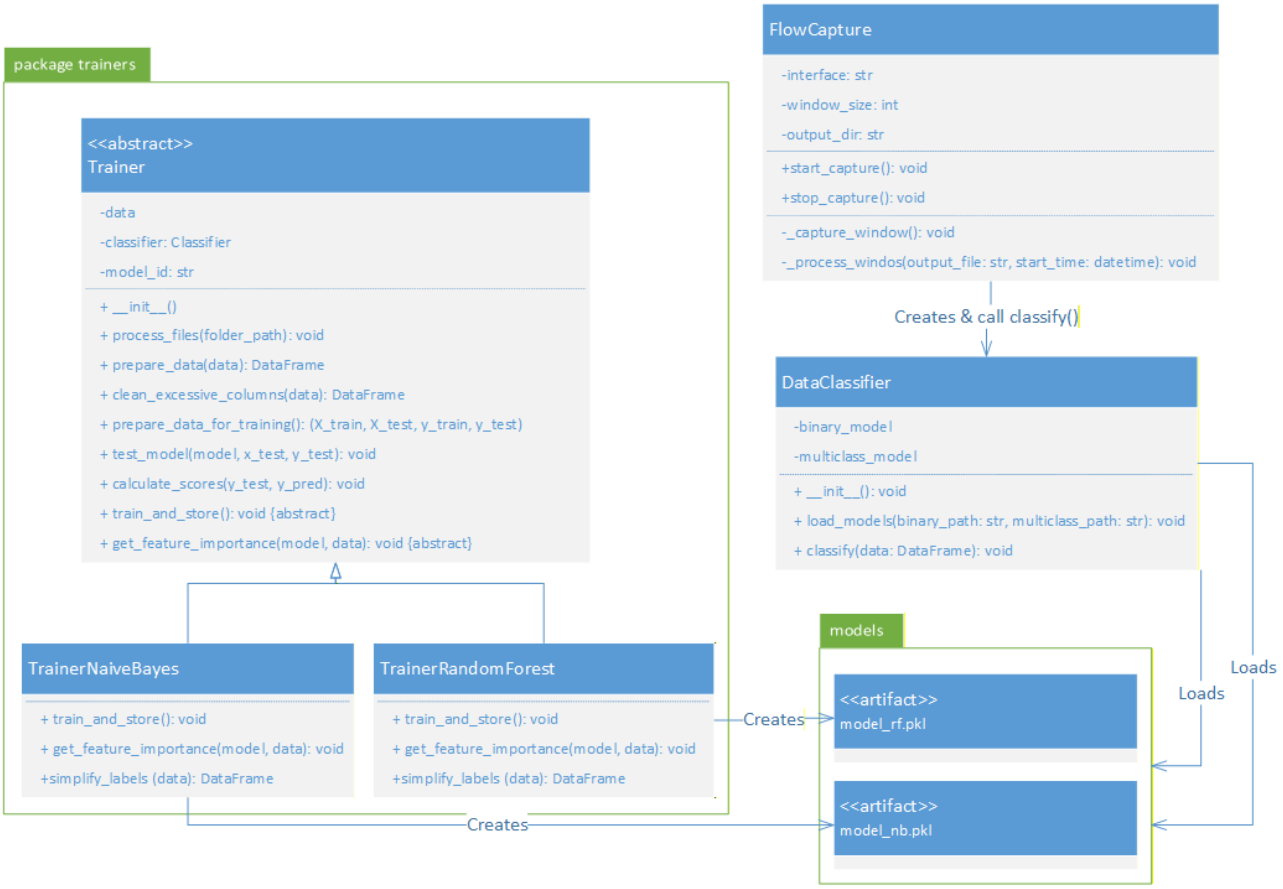

Рис. 3.5. Діаграма модульної архітектури навчання та застосування класифікаторів (*Джерело: сформовано автором)*

Для запуску процесу захоплення використовується метод start_capture(). Цей метод переводить систему в активний режим (self.running = True) і починає цикл захоплення, що триває поки self.running істинне (тобто доки не отримано сигнал зупинки або не виникла помилка). В цьому основному циклі послідовно збираються та обробляються вікна даних (див. рис. 3.6).

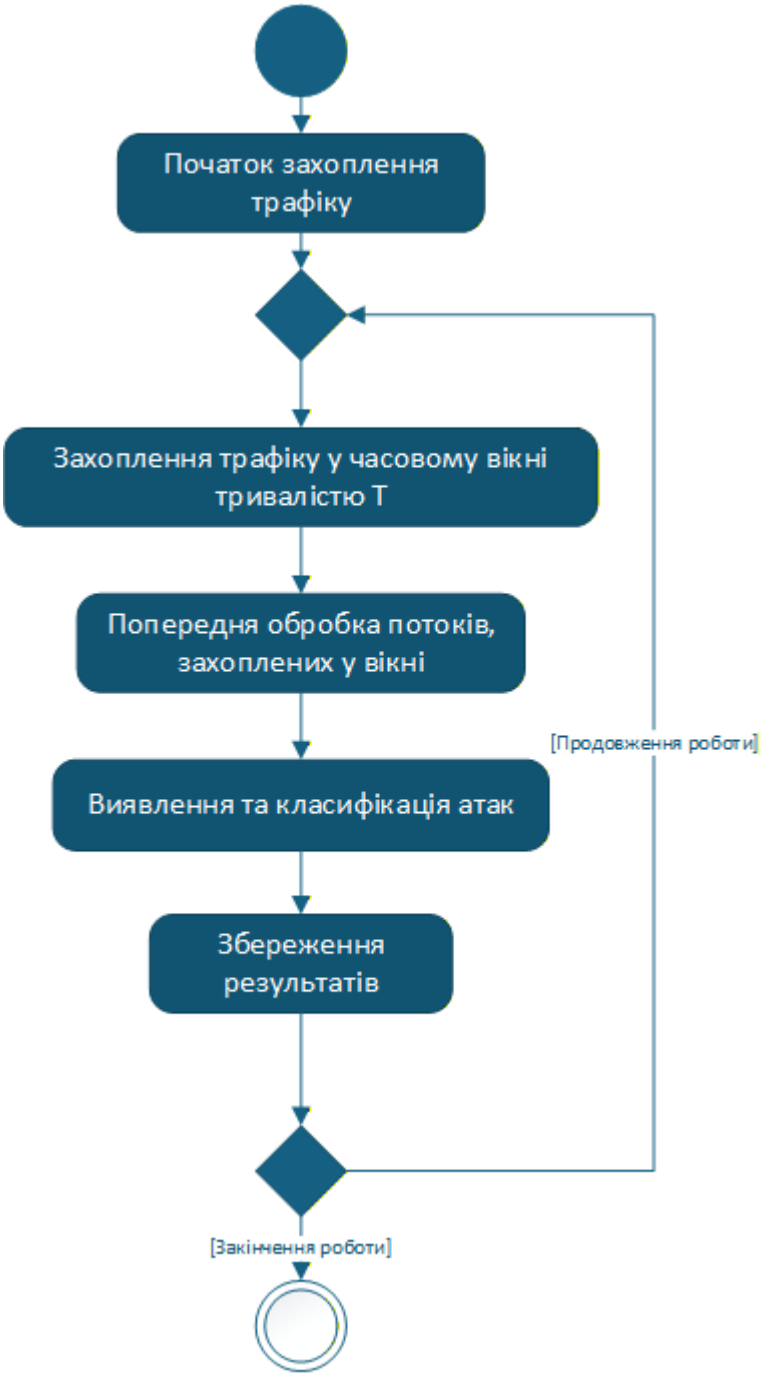

Рис 3.6. Діаграма діяльності алгоритму віконної обробки мережевого трафіку.

Джерело: розроблено автором на основі [25].

Основний цикл в start_capture() реалізовано як нескінченний while-цикл, який повторює захоплення даних з паузою між інтервалами. Кожна ітерація циклу відповідає одному часовому вікну захоплення. На початку ітерації запускається процес збору мережевих потоків за допомогою внутрішнього методу _capture_window(). Цей метод відповідає за захоплення мережевого трафіку із заданого інтерфейсу впродовж фіксованого інтервалу часу. В реалізації

використовується бібліотека PyFlowMeter (через виклик create_sniffer), що запускає сніффер – фонового збірника мережевих потоків – який накопичує статистику по кожному потоку (адреси, порти, протокол, лічильники пакетів/байтів тощо) і зберігає ці дані у файл CSV. Таким чином, протягом window_size секунд сніффер пасивно моніторить трафік на інтерфейсі і агрегує його у вигляді записів про потоки.

По завершенні інтервалу window_size захоплення для поточного вікна зупиняється, і зібрані за цей період потоки фіксуються у тимчасовому CSV-файлі (назва генерується з міткою часу початку або кінця вікна). Далі відбувається обробка вікна – викликається метод _process_window(output_file, start_time), що аналізує файл з потоками і запускає їх подальшу класифікацію (див. нижче). Після обробки на початку наступної ітерації робиться невелика пауза (близько 1 секунди), щоб уникнути накладання вікон, після чого (якщо захоплення не зупинено) запускається збір даних для нового вікна. Таким чином, цикл чергує фази збору даних та їх обробки з коротким інтервалом відпочинку. Якщо під час захоплення виникає виключення або помилка, цикл переривається і управління переходить до блока finally для коректного завершення роботи (див. рис. 3.7).

Клас FlowCapture передбачає коректне завершення циклу за сигналом. Метод stop_capture() зупиняє роботу сніффера: якщо об’єкт захоплення існує, викликається його метод stop() і join() для завершення потоку збору трафіку. Це гарантує, що поточне вікно завершується акуратно і всі дані записані, після чого виводиться повідомлення про зупинку захоплення. Логіка обробки сигналів убезпечує від втрати даних або зависання підчас примусового завершення роботи програми.

По закінченню кожного інтервалу захоплення, коли тимчасовий CSV-файл з потоками сформовано, запускається процедура його обробки та підготовки даних до класифікації. Ця процедура реалізована в методі _process_window(), який приймає шлях до згенерованого файлу та мітку часу початку вікна.

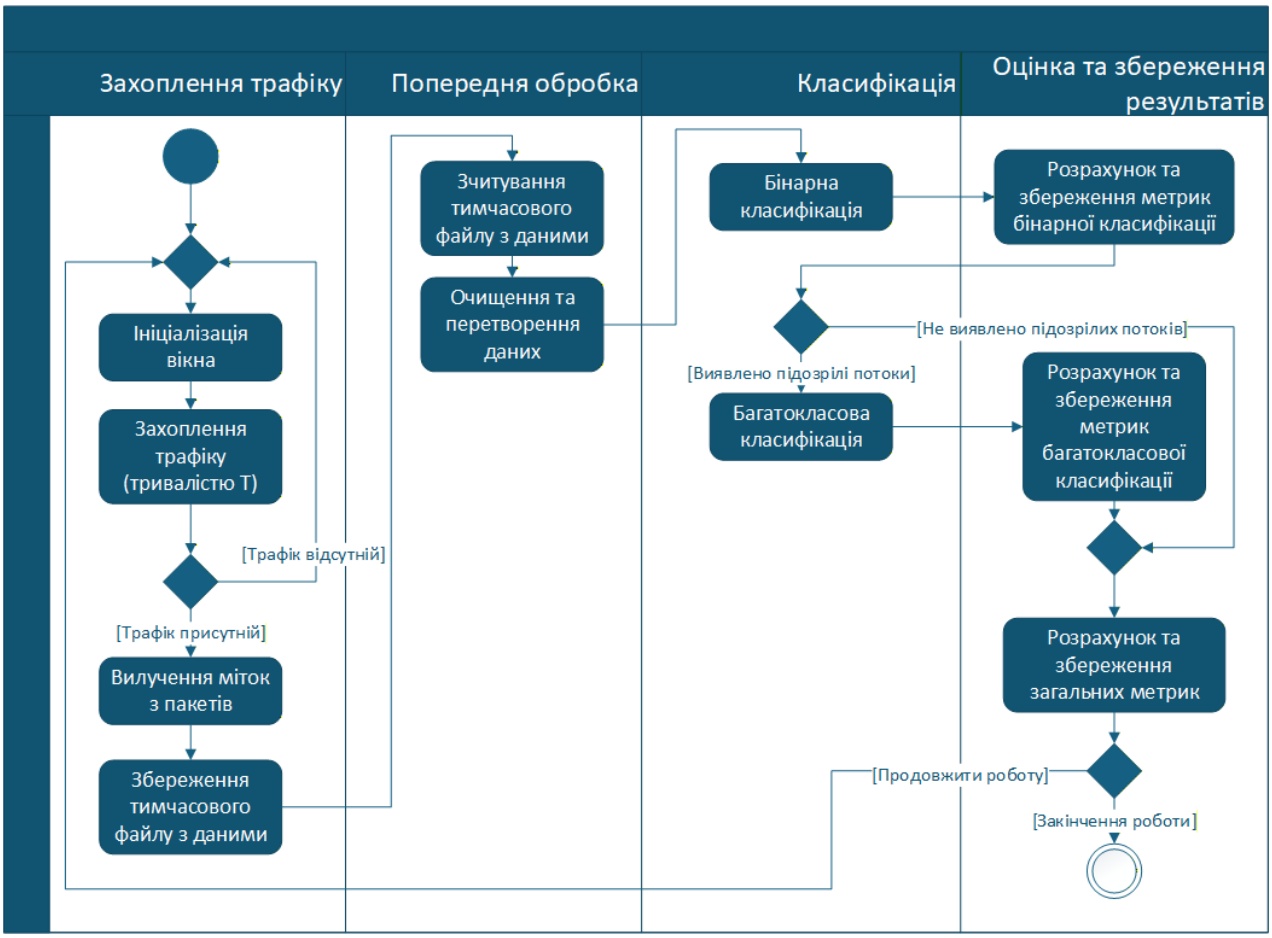

Рис. 3.7. Діаграма алгоритму обробки даних під час збору та класифікації трафіку (*Джерело: сформовано автором за* [27])

Зчитування даних. Першим кроком метод перевіряє наявність файлу з потоками для поточного вікна і те, що файл не порожній. Якщо дані існують, файл зчитується з диску у pandas DataFrame для подальшої обробки в пам'яті. Якщо файл відсутній або його розмір нульовий, то робиться висновок, що за цей інтервал не було зафіксовано жодного потоку; в такому разі обробка припиняється, в консоль виводиться повідомлення «No flows captured», і виконання переходить до наступного вікна без класифікації (метод _process_window просто збільшує лічильник вікон і завершується).

Формування Flow ID. Після успішного завантаження даних перевіряється наявність колонки ідентифікатора потоку (Flow ID). Якщо Flow ID відсутній серед стовпців (наприклад, сніффер не згенерував це значення), то він створюється вручну. Ідентифікатор генерується на основі атрибутів потоку: конкатенуються IP-адреса джерела та порт, IP-адреса і порт призначення, а також протокол, розділені розділовими знаками (формат «SrcIP:SrcPort-DstIP:DstPort-

Protocol»). Така комбінація унікально визначає потік і дозволяє відстежувати його в подальшому. Згенероване значення Flow ID додається як нова колонка до DataFrame; для зручності цю колонку вставляють на початку таблиці як перший стовпець.

Перейменування ознак. Формат вихідних полів, отриманих від PyFlowMeter, може відрізнятися від форматів набору даних, на яких навчено моделі. Тому здійснюється перейменування стовпців згідно з заздалегідь визначеним словником з константи COLUMNS_MAPPING. Це словник відповідності, що перейменовує поля DataFrame у стандартизований набір назв (наприклад, може змінити lowercase імена на формат із пробілами та великими літерами, що збігаються з оригінальними ознаками USB-IDS-1, або навпаки). Мета – привести назви стовпців до єдиного формату COLUMNS, який очікують моделі класифікації. Після перейменування виконується перевірка/відбір ознак: формується список available – ті імена з еталонного списку COLUMNS, які присутні в поточному фреймі даних. Можливо, деякі стовпці в захоплених даних відсутні або містять лише порожні значення, тому відбираються лише релевантні ознаки. Також на цьому етапі відсіюються службові колонки, які не є вхідними ознаками моделі (наприклад, IP-адреси, порти, Flow ID тощо вже або вилучені, або не включені до COLUMNS). Це відбувається для гармонізації даних із фрейму з класифікаційною моделлю. В результаті DataFrame містить тільки числові характеристики потоків, необхідні для роботи класифікатора.

Збереження очищених даних. Підготовлені дані поточного вікна (після генерації Flow ID, додання часу, перейменування та фільтрації ознак) зберігаються у вигляді CSV-файлу в цільовому вихідному каталозі. Ім’я файлу включає тимчасову мітку (наприклад, flows_window_YYYYMMDD_HHMMSS.csv), що дозволяє ідентифікувати, якому інтервалу належить файл. Цей файл містить вже очищені, стандартизовані показники по кожному зафіксованому потоку і може використовуватись для подальшого офлайн-аналізу чи навчання моделей. Важливо, що на момент збереження цього файлу класифікація ще не виконана – файл містить фактичні виміряні ознаки потоків, але не результати класифікації (мітки класів).

Класифікація виконується наступним кроком у пам’яті, безпосередньо над DataFrame.

Ініціалізація класифікатора. Після успішного формування набору ознак, FlowCapture переходить до етапу класифікації потоків. Для цього створюється екземпляр класу DataClassifier. При ініціалізації цього об’єкта відбувається завантаження заздалегідь навчених моделей машинного навчання: з диску зчитується бінарна модель (для класифікації «нормальний/атакувальний» трафік) та багатокласова модель (для розпізнавання типу атаки серед шкідливих потоків). Шляхи до файлів моделей задаються конфігурацією (константи Config.BINARY_MODEL_PATH і Config.MULTICLASS_MODEL_PATH) і передаються конструктору DataClassifier. Відповідні моделі завантажуються через утиліту pickle (відбувається десеріалізація з файлу) – функція load_model абстрагує цей процес, повертаючи готовий до використання об’єкт моделі. На цьому етапі класифікатор готовий приймати підготовлені дані потоків для прогнозування.

Етап класифікації здійснюється методом classify() об’єкта DataClassifier, якому передають DataFrame з ознаками потоків поточного вікна та мітку часу (ідентифікатор сесії). Усередині цього методу реалізовано декілька підпроцесів: передобробка даних для моделі, двофазна класифікація (бінарна та багатокласова) та оцінка результатів. Фактично це така ж сама операція, що здійснюється після навчання моделі у тренері, але вона виконується для щойно зібраних мережевих даних з порту, а не для навчального набору.

Бінарна класифікація. На першому підетапі класифікації DataClassifier застосовує завантажену бінарну модель до підготовлених даних потоків. Модель вирішує задачу двокласового розподілу: визначає, які потоки є нормальними (легітимний/безпечний трафік), а які мають ознаки атаки/загрози. Цей етап моделі розраховує імовірності приналежності до класу «нормальний» чи «загроза» для кожного зразка (наприклад, probability_safe та probability_threat). Після виконання прогнозування для всіх потоків у вікні, до DataFrame додається колонка predicted (прогнозований клас). Вона розраховується на основі порогу

спрацьовування, тобто якщо ймовірність атаки вища порогу, або ймовірність нормального трафіку нижча порогу, то такий потік розпізнається як загроза.

Багатокласова класифікація. На другому підетапі класифікації DataClassifier задіює багатокласову модель, але лише для підмножини даних – тих потоків, які були позначені бінарною моделлю як «загроза». Під час класифікації для інтерпретації прогнозу використовується наступна логіка: багатокласовий класифікатор повертає індекс або мітку прогнозованого класу атаки, яка відповідає одній з відомих категорій. Цей результат заноситься до датафрейму (attack для атаки та confidence для ймовірності). У підсумку, після двофазної обробки, кожен мережевий потік у DataFrame матиме класифікаційну мітку: «нормативний» (безпечний) або конкретний тип атаки (для шкідливих потоків).

Оцінка якості класифікації. Якщо захоплені дані містять відомі еталонні мітки класів (наприклад, якщо трафік генерується в контрольованому середовищі з наперед відомим «правильним» класифікаційним ярликом для кожного потоку), то система виконує оцінювання точності роботи моделі на цьому вікні. В DataClassifier.classify() передбачено збір реальних міток y_test та прогнозованих y_pred для поточного набору потоків і обчислення набору стандартних метрик класифікації. Використовуються метрики макроусередненої точності (accuracy), точності класифікації (precision), повноти (recall) та F_1-міри, а також формується матриця невідповідностей (confusion matrix), що показує розподіл правильних/помилкових розпізнавань по кожному класу. Обчислення виконується за допомогою функцій із бібліотеки scikit-learn (наприклад, accuracy_score, precision_score тощо) на основі масивів істинних і передбачених класів. Розраховані показники разом із службовою інформацією передаються у функцію check_scores для логування.

Утилітна функція check_scores(...) зберігає зведену інформацію про якість класифікації в загальний файл результатів (CSV-таблицю). Кожен виклик формує один рядок у цьому файлі, що відповідає одному аналізованому вікну (або одній фазі класифікації). Відомості, що заносяться: часова позначка (timestamp – час обробки вікна), тип класифікації (classification – наприклад, «binary» чи

«multiclass», щоб розрізняти двофазні результати), кількість потоків у вибірці (number_of_flows), варіативність потоків (flows_variance) – у цьому полі зберігається список унікальних класів, присутніх серед розглянутих потоків (наприклад, які типи атак спостерігались у даному вікні), матриця невідповідностей (confusion_matrix) у вигляді списку, а також значення метрик точності, точності класифікації, повноти і F_1.

Окрім збереження у файл, результати класифікації виводяться на екран для контролю. Зокрема, DataFrame із доданими прогнозами (flows) відображається у консолі, що дозволяє оператору побачити, скільки потоків і яких було виявлено, та з якими мітками. Також виводяться повідомлення про завершення обробки чергового вікна, із зазначенням його номера і об’єму даних. Це забезпечує зворотний зв’язок у реальному часі про роботу алгоритму.

Алгоритм FlowCapture приділяє особливу увагу збереженню усіх проміжних та фінальних результатів, щоб забезпечити можливість подальшого аналізу та відтворюваність. По-перше, кожне часове вікно генерує файл потоків CSV у вихідній директорії, що містить очищені та підготовлені дані потоків цього інтервалу. Ці файли акумулюють вихідні ознаки (feature values) потоків і можуть слугувати набором даних для додаткового навчання або перевірки моделей на більших обсягах (зверніть увагу: вони не містять колонок з результатами класифікації, лише виміряні параметри та, за потреби, Flow ID і часову мітку). По-друге, у файлі з результатами класифікації (визначеному в Config.FILE_RESULTS) накопичуються метрики роботи моделі для кожного вікна або для кожного етапу класифікації. Це дає можливість проаналізувати ефективність в динаміці – наприклад, як змінюється точність виявлення атак з часом або в залежності від навантаження мережі.

В кінці сеансу захоплення або при отриманні сигналу зупинки, клас FlowCapture забезпечує коректне завершення захоплення. Метод stop_capture() зупиняє сніффер і завершує його потік виконання, що гарантує запис останнього неповного вікна (якщо такий є) і вивільнення ресурсів. Програма виводить повідомлення «Capture stopped.» і припиняє основний цикл. Всі зібрані дані та обчислені результати залишаються збереженими у відповідних файлах. Таким

чином, навіть у випадку примусового переривання, алгоритм не втрачає вже накопиченої інформації. Це важливо для цілісності експериментальних даних та подальшого використання результатів класифікації в інших модулях або звітах.

3.4. Аналіз та обговорення результатів

У цьому розділі представлено аналіз результатів, отриманих у процесі реалізації та тестування дворівневої системи виявлення атак у мережевому трафіку. Метою аналізу є оцінка ефективності побудованої моделі з точки зору точності класифікації, стабільності роботи в різних умовах і стійкості до зсуву домену (domain shift) між різними джерелами даних.

Для кількісної оцінки статистичного зсуву між навчальним та цільовим середовищами було виконано порівняльний аналіз інтегральної міри розбіжності S_{domain}, визначеної в підрозділі 2.4. З метою відокремлення варіативності даних від міждоменного ефекту спочатку оцінено внутрішньодоменну нестабільність.

Кожен із досліджуваних датасетів було поділено на дві неперекривні частини у співвідношенні 50/50 без випадкового перемішування, що дозволяє зберегти часову структуру трафіку. Для кожної пари підвибірок обчислено значення інтегральної розбіжності S_{domain}.

Для набору USB-IDS-1 отримано $S_{USB\_in} = 1.5927$, що відображає базовий рівень статистичної неоднорідності в межах одного домену. Для власного експериментального середовища значення склало: $S_{OWN\_in} = 2.8350$.

Отримані величини характеризують внутрішньодоменний розкид, зумовлений зміною інтенсивності потоків, варіативністю ознак та скінченністю вибірки. Таким чином, навіть у межах одного домену спостерігається ненульовий рівень статистичних коливань.

Аналогічну процедуру було застосовано для повних наборів USB-IDS-1 та власного середовища. У цьому випадку отримано $S_{cross} = 11.3423$.

Це значення суттєво перевищує внутрішньодоменні оцінки.

Для кількісного зіставлення було обчислено відносні коефіцієнти перевищення:

$$\frac{S_{cross}}{S_{USB\_in}} \approx 7.1, \frac{S_{cross}}{S_{OWN\_in}} \approx 4.0.$$

Отже, інтегральна розбіжність між доменами перевищує природну внутрішньодоменну варіативність у 4–7 разів. Це свідчить про наявність вираженого статистичного зсуву розподілів ознак між навчальним і цільовим середовищами.

Слід зазначити, що оцінювання виконувалося на повних наборах даних, включно з нормативним трафіком та всіма типами атак. У контексті практичного застосування IDS така інтегральна оцінка є доцільною, оскільки система функціонує в умовах зміни як статистичних характеристик трафіку, так і співвідношення між класами. Отримані результати підтверджують, що деградація метрик класифікації при переході між доменами є статистично обґрунтованою та пов'язана з істотною зміною структури даних, а не з випадковими коливаннями або нестабільністю моделі.

З метою емпіричної перевірки коректності реалізації порогового механізму першого рівня, аналітично обґрунтованого у підрозділі 2.3, було проведено серію обчислювальних експериментів зі зміною параметра вартості пропуску атаки C_{FN} за фіксованого значення C_{FP}. Оскільки рішення першого рівня приймається у просторі log-odds, варіювання параметра ризику безпосередньо впливає на порогове значення τ_b, що використовується під час класифікації, а отже – на частку потоків, переданих на другий рівень аналізу. Для кількісної оцінки цього ефекту використано показник gate_rate – частку потоків, які першим рівнем віднесено до підозрілих.

На рисунку 3.8 наведено результати для випадку узгоджених навчальної та тестової вибірок (модель навчена і протестована на власних даних). Зі зростанням C_{FN} порогове значення τ_b у log-odds просторі монотонно зменшується, переходячи з додатної області у від'ємну. Одночасно спостерігається систематичне зростання показника gate_rate приблизно з 0.58 до 0.75. Така залежність підтверджує коректність напрямку реакції порогового правила: зменшення порогу призводить до збільшення кількості потоків, що класифікуються як потенційно підозрілі.

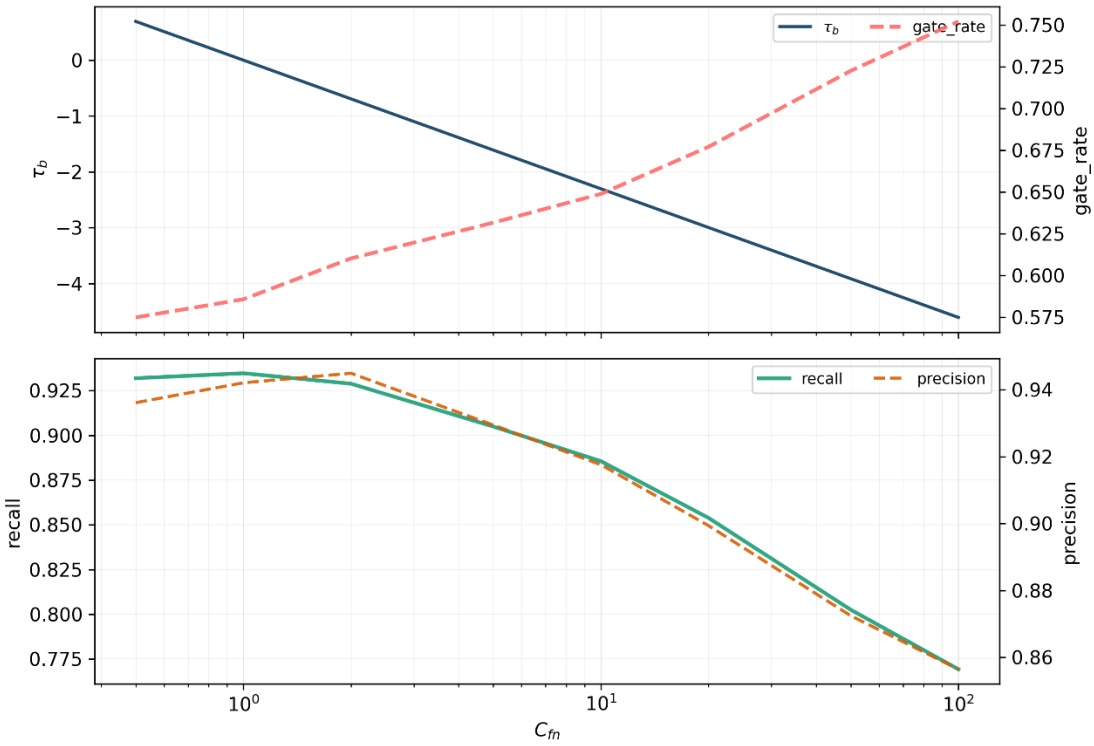

Рис. 3.8 – Залежність порогу τ_b, коефіцієнта каскадної передачі (gate_rate) та метрик якості (recall, precision) від вартості пропуску атаки C_{FN} (модель навчена і протестована на власних даних) (*Джерело:сформовано автором)*

Вплив зміни порогу відображається і в поведінці метрик першого рівня. У межах sweep-експерименту показник recall знижується приблизно з 0.93 до 0.77, тоді як precision зменшується з 0.94 до 0.86. Таким чином, варіюванняC_{FN} забезпечує керовану зміну політики прийняття рішень: при зменшенні порогу система передає більше потоків на другий рівень, що супроводжується зміною співвідношення між чутливістю та точністю. Для узгодженого домену перший рівень демонструє виражену та передбачувану реакцію на зміну параметра вартості помилки.

На рисунку 3.9 подано результати для випадку невідповідності навчальної та тестової вибірок (модель навчена на власних даних і протестована на USB-IDS-1).

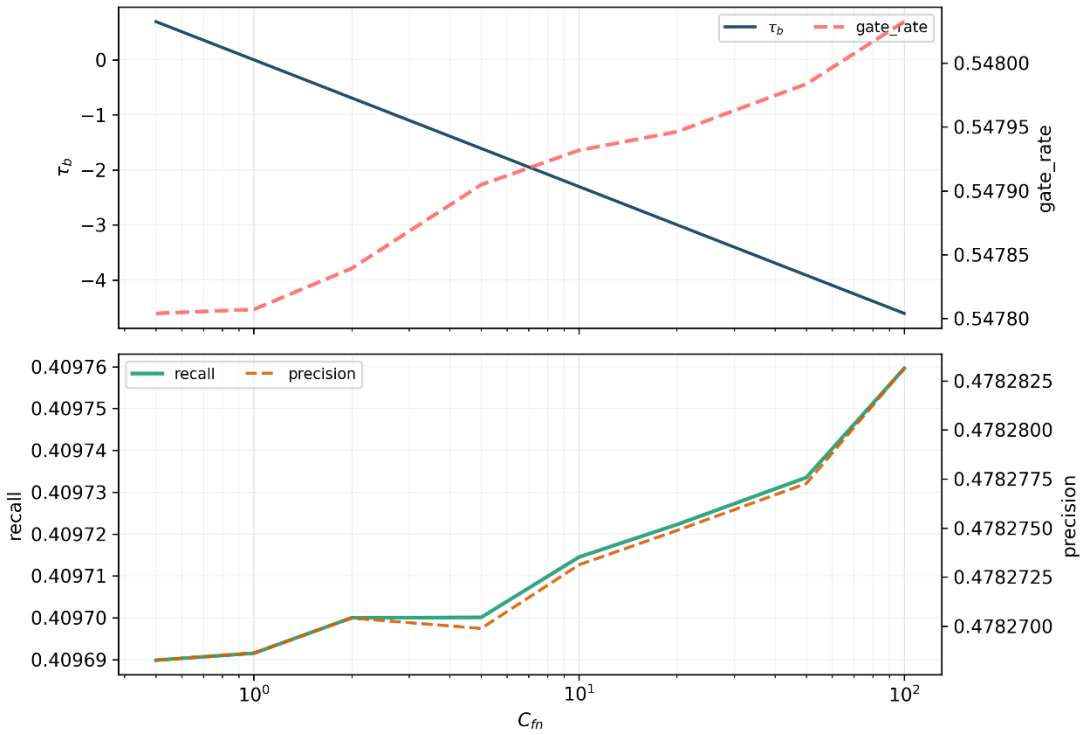

Рис. 3.9 – Залежність порогу τ_b, коефіцієнта каскадної передачі (gate_rate) та метрик якості (recall, precision) від вартості пропуску атаки C_{FN} (модель навчена на власних даних і протестована на USB-IDS-1) (*Джерело:сформовано автором)*

У цьому випадку формальна монотонність залежності $\tau_b(C_{FN})$ та gate_rate зберігається, однак амплітуда змін є істотно меншою: показник gate_rate змінюється лише в межах приблизно 10^{-4}, а значення recall та precision залишаються практично незмінними. Це свідчить про низьку чутливість порогового механізму до варіювання C_{FN} за умов доменного зсуву. Фактично зміна порогу майже не впливає на емпіричні характеристики моделі, що вказує на обмеженість універсальності статичних порогових налаштувань при зміні статистичної структури трафіку.

Аналогічна картина спостерігається для моделі навченої на наборі USB-IDS-1 і протестовано на власних даних, де зміна C_{FN} також практично не впливає на значення recall та precision. Натомість у випадку узгодженого домену USB-IDS-1 зберігається передбачувана реакція системи на зміну порогу, хоча амплітуда змін є меншою порівняно з випадком домену власних даних. Це може бути пов’язано з особливостями розподілу log-odds для відповідного набору даних.

Узагальнюючи результати аналізу, можна стверджувати, що перший рівень запропонованої інформаційної технології коректно реалізує аналітично обґрунтований механізм мінімізації ризику та забезпечує керовану зміну поведінки системи через параметр C_{FN}. Водночас ефективність порогового налаштування істотно залежить від статистичної узгодженості навчального та тестового доменів: у випадку доменного зсуву вплив зміни C_{FN} на емпіричні метрики стає практично незначним. Це обґрунтовує необхідність використання механізмів оцінювання доменного зсуву та адаптивної корекції параметрів на наступному рівні системи.

Онлайн-перевірка роботи моделі

Аналіз проводився на двох наборах даних – публічному USB-IDS-1 та власному емульованому трафіку, зібраному у віртуальному середовищі GNS3. Такий підхід дозволяє оцінити узагальнювальну здатність моделі та визначити вплив відмінностей у структурі потоків, топології мережі та фонових навантаженнях на результати класифікації. Обробка понад 6 млн та 5 млн пакетів здійснювалася у вигляді послідовності 60-секундних часових вікон, що дозволяє аналізувати зміну метрик не точково, а у динаміці. Кожне вікно містило від сотень до тисяч мережевих потоків, сформованих з різною інтенсивністю трафіку та типами взаємодій.

Вибір тривалості вікна агрегації у 60 секунд обумовлений необхідністю забезпечення балансу між швидкістю виявлення атак та статистичною стійкістю оцінюваних показників. Зменшення інтервалу до 10 секунд призвело б до суттєвого зменшення кількості потоків у межах одного вікна, що, у свою чергу, спричинило б зростання стохастичної варіативності метрик та нестабільність оцінки інтегральної міри зсуву домену S_{domain}. За умов малої вибірки розподіл ознак може змінюватися випадковим чином, що ускладнює розмежування природних флуктуацій трафіку та реального статистичного зсуву. Водночас надмірне збільшення тривалості інтервалу, наприклад до 5 хвилин, призводить до згладжування локальних змін у структурі мережевих потоків та зменшення часової роздільної здатності системи. У такому випадку реакція на появу нових

або короткочасних аномалій відтерміновується, що негативно впливає на оперативність детекції.

Інтегральна міра зсуву S_{domain}є функцією емпіричного розподілу ознак у межах конкретного інтервалу спостереження. За надто коротких вікон оцінка розподілу втрачає репрезентативність, що може спричиняти штучні коливання значення S_{domain}і, відповідно, некоректну активацію адаптивного порогового механізму. За надто довгих вікон, навпаки, відбувається згладжування різких змін статистичного профілю трафіку, що зменшує чутливість системи до швидких атак або коротких сплесків активності. Обраний 60-секундний інтервал забезпечує достатній обсяг потоків для стабільної оцінки розподілів ознак та дозволяє відокремити внутрішньодоменну варіативність від суттєвого міждоменного зсуву, зберігаючи при цьому адекватну часову реакцію системи.

Після проведення емуляційних та класифікаційних експериментів було зібрано набір мережевих даних, які були згенеровані за допомогою атакувальних скриптів та генератора нормального трафіку. Даний набір містить 41 850 записів даних про потоки та відмінну структуру даних як за значеннями, так і за розподілом від набору USB-IDS-1.

У структурі даних у різних наборах даних спостерігається відмінність, яка є ознакою domain/topology shift – оскільки власні дані збиралась в інших мережевих умовах, ніж набір USB-IDS-1, статистичні дані потоків мають помітну різницю в значеннях, наприклад для таких ознак як Flow duration, Packet length variance, а також Fwd IAT Mean/Std/Max/Min. Також така різниця пов'язана з різними характеристиками емульованих атак, та даних про атаки зібрані в наборі USB-IDS-1 (рис. 3.10 та 3.11).

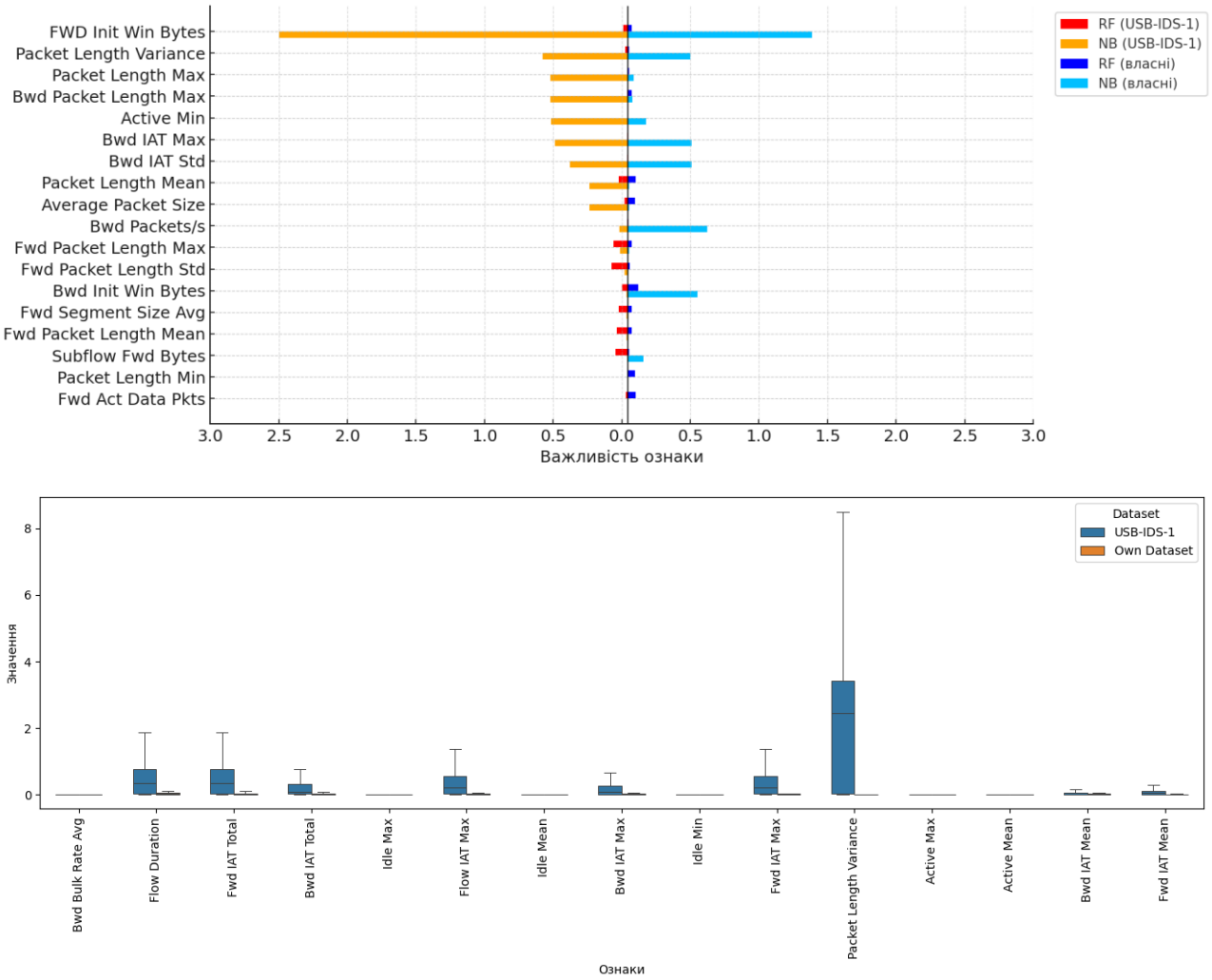

Рис. 3.10. Порівняння розподілу ознак між наборами даних

(*Джерело: сформовано автором)*

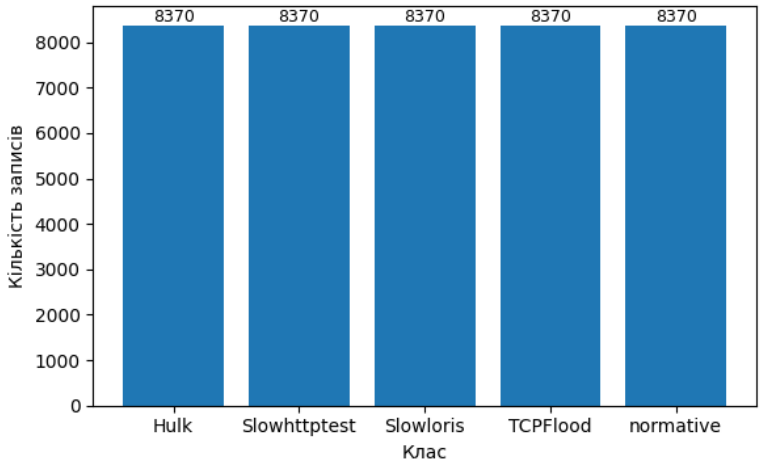

Рис. 3.11. Розподіл даних за класами у власному наборі даних

(*Джерело: сформовано автором)*

Розглянемо значення метрик ефективності для дворівневої моделі NB/RF у випадку in-domain (модель протестована на даних, подібних до тих, які були

використані для навчання), cross-domain (навчена на наборі USB-IDS-1) та mixed domain (навчена на змішаному наборі даних).

У режимі in-domain , коли навчальна та тестова вибірки належать до одного статистичного середовища, результати віконного аналізу демонструють важливу закономірність: навіть за наявності помітних коливань інтегральної міри зсуву домену S_{domain}показники ефективності дворівневої моделі залишаються стабільними та високими (рис. 3.12). Червона крива S_{domain}, представлена на верхньому графіку, свідчить про те, що розподіл ознак у межах одного домену не є строго стаціонарним. Між окремими часовими вікнами спостерігаються флуктуації, зумовлені зміною інтенсивності трафіку, варіаціями частки атак, різною кількістю сформованих потоків та природною стохастичною мінливістю мережевих характеристик. Таким чином, навіть у контрольованому середовищі присутня внутрішньодоменна статистична неоднорідність.

Водночас ці коливання S_{domain} не супроводжуються пропорційним погіршенням метрик класифікації. Значення Accuracy упродовж аналізованих вікон утримуються на рівні близько 0.97, а F1 – у межах 0.87–0.89, при відносно невеликих стандартних відхиленнях. Відсутність різких провалів показників навіть у моменти пікових значень S_{domain}свідчить про те, що внутрішньодоменні зміни не перевищують меж, у межах яких модель здатна зберігати коректну геометрію розділяючих поверхонь у просторі ознак. Інакше кажучи, сформована під час навчання структура ансамблю Random Forest є достатньо робастною до природних варіацій трафіку, що виникають у межах того самого середовища.

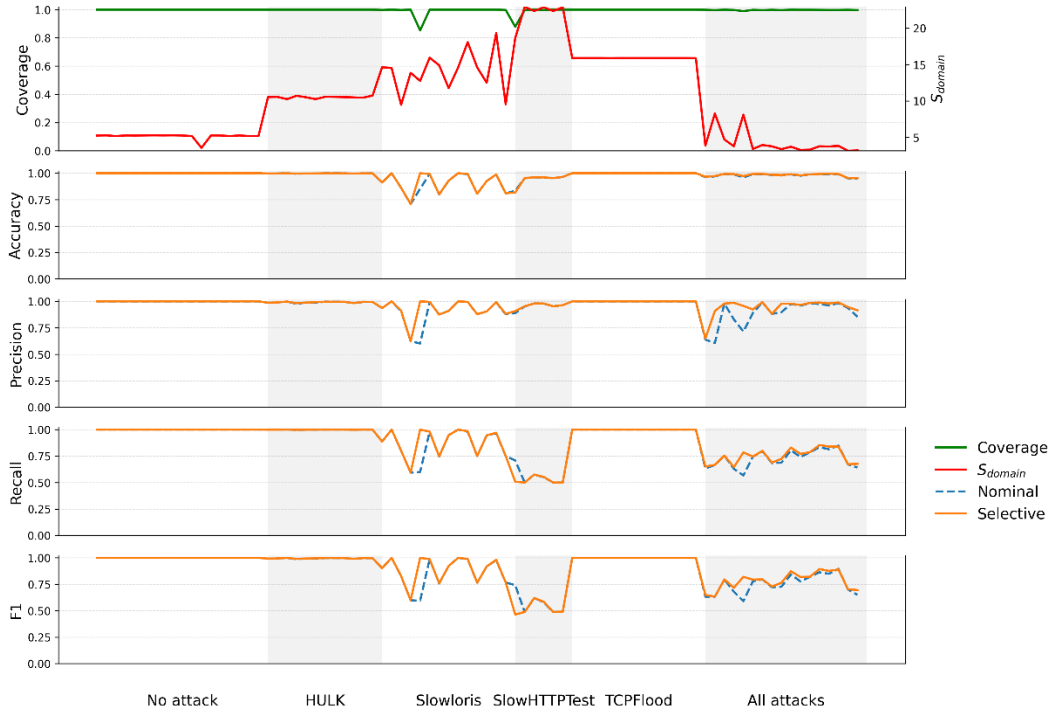

Рис. 3.12. Віконна оцінка ефективності дворівневої моделі прийняття рішень у різних сценаріях мережевого трафіку (in domain) (*Джерело:сформовано автором)*

Аналіз співвідношення precision та recall у in-domain режимі підтверджує збалансованість роботи системи. Значення precision близькі до 0.95, а recall – до 0.88, що означає ефективне виявлення більшості атак при контрольованому рівні хибнопозитивних спрацювань. Невеликі коливання цих метрик між вікнами мають випадковий характер і не формують довготривалих негативних трендів. Це свідчить про відсутність накопичення систематичної помилки або поступової деградації моделі у часі. Таким чином, внутрішньодоменна варіативність відображає лише природний шум, а не структурну невідповідність моделі даним. Перехід до селективного режиму в in-domain умовах практично не змінює середні значення Accuracy та F1, однак істотно знижує coverage (приблизно з 0.99 до 0.53). Це означає, що механізм відмови активується навіть за помірних флуктуацій S_{domain}, проте не дає суттєвого приросту якості, оскільки модель і без того працює у близькому до оптимального режимі. Зниження coverage за відсутності помітного покращення метрик свідчить про те, що у статистично узгодженому домені додаткова селективність не є критично необхідною. Отже, in-domain графік демонструє не відсутність зсуву як такого, а здатність запропонованої дворівневої архітектури витримувати внутрішньодоменну

статистичну мінливість без істотної втрати якості класифікації. Це створює базовий еталон стабільності, з яким надалі коректно порівнювати поведінку системи у cross-domain та mixed-domain режимах.

У cross-domain режимі графік демонструє принципово іншу картину порівняно з in-domain (рис. 3.13). Інтегральна міра зсуву домену S_{domain} має підвищений та нестабільний рівень у більшості сегментів, що відповідають атакам (HULK, Slowloris, SlowHTTPTest, TCPFlood).

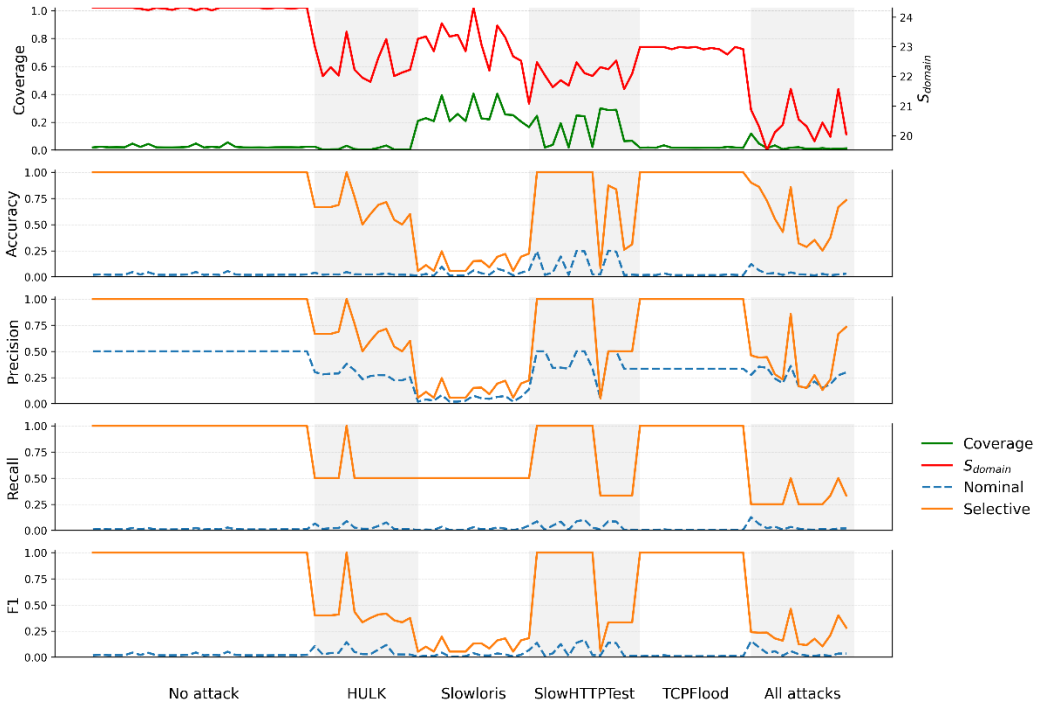

Рис. 3.13. Віконна оцінка ефективності дворівневої моделі прийняття рішень у різних сценаріях мережевого трафіку (cross domain)

(*Джерело: сформовано автором)*

На відміну від in-domain випадку, де коливання S_{domain}не призводили до суттєвої деградації метрик, тут спостерігається чітка кореляція між зростанням S_{domain}та падінням якості класифікації. Це означає, що статистичні розбіжності між навчальним та цільовим розподілами перевищують допустимий рівень внутрішньодоменної варіативності, і модель втрачає узгодженість із новим середовищем.

У номінальному режимі (без адаптивного порогу) показники Accuracy, Precision, Recall та F1 у більшості вікон перебувають на низькому рівні. Особливо це видно в сегментах, де S_{domain} набуває максимальних значень: метрики або

різко знижуються, або залишаються близькими до нуля. Така поведінка свідчить про системну помилковість рішень другого рівня. Модель продовжує примусово атрибутувати потоки до одного з відомих класів атак, навіть коли розподіл ознак істотно відрізняється від навчального. У результаті виникає масова хибна атрибуція, що призводить до майже повної втрати F1 у окремих сегментах. Важливо, що при цьому варіативність nominal-метрик відносно невелика – це означає, що деградація не випадкова, а структурна.

Селективний режим демонструє іншу динаміку: у тих самих сегментах, де S_{domain} зростає, модель починає частіше переходити у стан відмови від рішення (що відображається у зниженні coverage). Завдяки цьому різко зменшується кількість хибних багатокласових атрибуцій, а показники Precision і F1 зростають порівняно з номінальним режимом. При цьому Recall може демонструвати більш складну поведінку: у деяких вікнах він зменшується через відмову від автоматичного рішення, проте сумарний ризик помилкової класифікації знижується. Таким чином, селективний режим не «виправляє» модель у класичному сенсі, а змінює політику прийняття рішень, перерозподіляючи помилки типу misclassification у контрольований клас Unknown.

Порівняння поведінки nominal і selective кривих у відношенні до S_{domain} дозволяє зробити принциповий висновок. У сегментах із підвищеним S_{domain} номінальний режим продовжує генерувати помилкові рішення з низькою якістю, тоді як селективний режим знижує coverage, але стабілізує метрики для тих потоків, які залишаються в автоматичній обробці. Отже, графік cross-domain наочно демонструє механізм компенсації доменного зсуву: при зростанні статистичної розбіжності система переходить до більш обережної стратегії, зменшуючи частку автоматичних атрибуцій і підвищуючи надійність прийнятих рішень. Це підтверджує коректність побудованої порогової логіки та її функціональну роль у дворівневій архітектурі IDS.

У mixed-domain режимі спостерігається проміжний характер статистичної узгодженості між навчальним і цільовим середовищами, що безпосередньо

відображається як у поведінці інтегральної міри зсуву S_{domain}, так і в динаміці метрик класифікації (рис. 3.14).

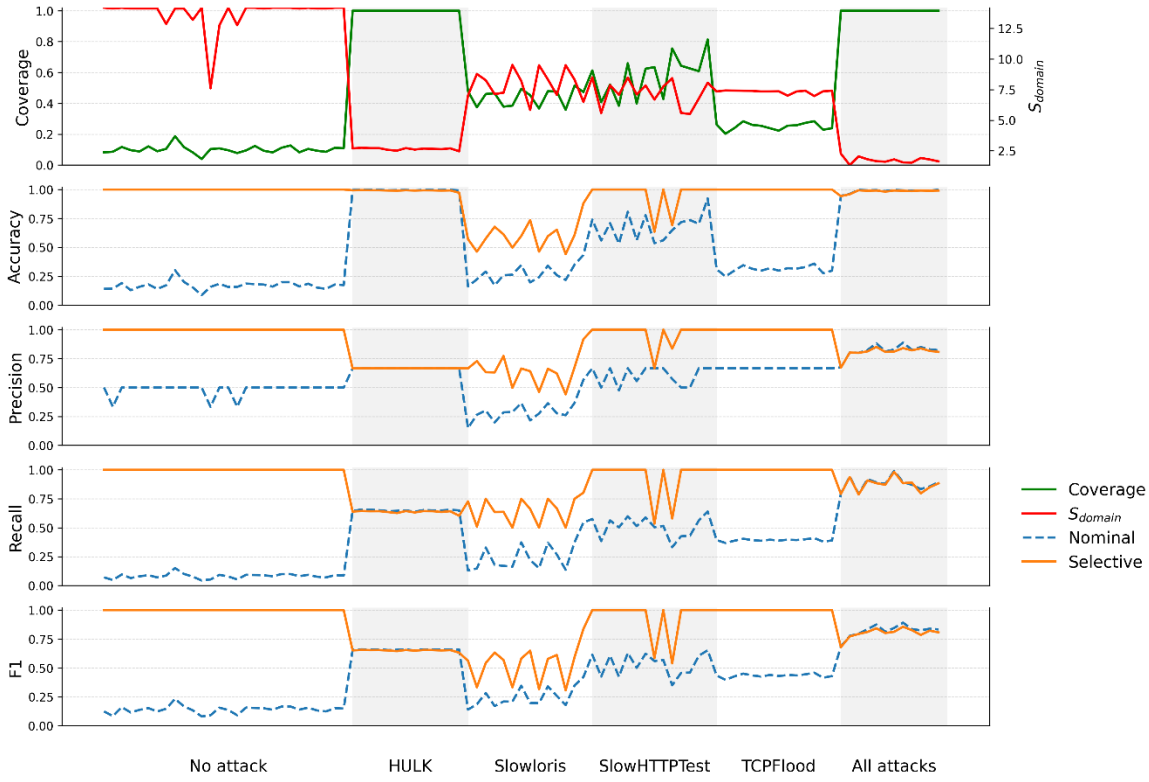

Рис. 3.14. Віконна оцінка ефективності дворівневої моделі прийняття рішень у різних сценаріях мережевого трафіку (mixed domain)

(*Джерело: сформовано автором)*

На відміну від in-domain сценарію, де коливання S_{domain} не призводили до суттєвої деградації показників, та від cross-domain режиму, де зростання S_{domain} супроводжувалося різким падінням якості, у змішаному середовищі спостерігається часткове накладання статистичних профілів. Це означає, що частина потоків зберігає ознаки, близькі до навчального розподілу, тоді як інша частина формує нові комбінації характеристик, що не були представлені у навчальній вибірці.

У номінальному режимі класифікації така неоднорідність проявляється у помірних середніх значеннях метрик (Accuracy ≈ 0.49, F1 ≈ 0.41) та значній варіативності між часовими вікнами. Наявність підвищених значень стандартного відхилення (σ ≈ 0.34–0.37 для Accuracy та σ ≈ 0.24–0.32 для F1) свідчить про нестабільність прийняття рішень у межах різних сегментів трафіку. Вікна з підвищеним S_{domain} корелюють зі зниженням precision і recall, що вказує

на часткову втрату узагальнювальної здатності моделі при відхиленні статистичного профілю ознак від навчального.

Перехід до селективного режиму призводить до суттєвого покращення агрегованих показників (Accuracy ≈ 0.68, F1 ≈ 0.62), що супроводжується зменшенням coverage. Це означає, що система обмежує автоматичну атрибуцію лише тими потоками, для яких впевненість класифікатора перевищує встановлений поріг. У сегментах із підвищеним S_{domain} coverage знижується, що запобігає масовій хибній класифікації та зменшує ризик некоректної атрибуції типів атак. Таким чином, селективний механізм виконує роль адаптивного фільтра, який стабілізує якість прийнятих рішень в умовах часткової статистичної невідповідності.

Отже, результати mixed-domain режиму підтверджують, що навіть помірний доменний зсув може призводити до нестабільності метрик у номінальному режимі, однак інтеграція механізму відмови дозволяє компенсувати негативний вплив зміни розподілів шляхом контрольованого зниження coverage. Це свідчить про доцільність використання адаптивної порогової політики в дворівневій архітектурі IDS для забезпечення більш надійної роботи в неоднорідних та змінних середовищах.

Аналіз агрегованих метрик

З метою узагальнення результатів віконного аналізу було виконано статистичну агрегацію метрик, обчислених окремо для кожного 60-секундного інтервалу. Значення агрегованих метрик наведено у таблиці 3.3. Для кожного домену та режиму класифікації сформовано вибірку значень m_i, де m_i– значення відповідної метрики (Accuracy, Precision, Recall, F1, Coverage) для і-го вікна, $i = 1, \dots, K$. Агреговані показники визначалися як середнє арифметичне по вікнах, а також обчислювалися мінімальне та максимальне значення (діапазон) і стандартне відхилення. Таким чином, статистичною одиницею аналізу виступало окреме часове вікно, що дозволяє оцінити не лише інтегральний рівень якості, але й її варіативність у часі та стабільність роботи моделі в різних експлуатаційних режимах.

Таблиця 3.3.

Агреговані оцінки ефективності роботи моделі[4]

Метрики	Домен	In-domain		Cross-domain		Mixed domain	
	Режим	**Номінал.**	**Селектив.**	**Номінал.**	**Селектив.**	**Номінал.**	**Селектив.**
Accuracy	Середнє	0,97	0,97	0,04	0,38	0,49	0,68
	Діапазон	0,70-1,00	0,70-1,00	0,01-1,00	0,002-1,00	0,09-0,99	0,04-1,00
	Стандартне відхилення	0,055	0,057	0,054	0,042	0,343	0,367
Precision	Середнє	0,95	0,93	0,32	0,53	0,57	0,73
	Діапазон	0,60-1,00	0,47-1,00	0,02-0,50	0,03-1,00	0,15-0,88	0,22-1,00
	Стандартне відхилення	0,092	0,115	0,161	0,329	0,165	0,202
Recall	Середнє	0,88	0,86	0,21	0,37	0,39	0,62
	Діапазон	0,50-1,00	0,33-1,00	0,004-0,12	0,004-1,00	0,04-0,99	0,019-1,00
	Стандартне відхилення	0,162	0,155	0,025	0,414	0,277	0,344
F1	Середнє	0,89	0,87	0,04	0,33	0,41	0,62
	Діапазон	0,48-1,00	0,30-1,00	0,006-0,16	0,007-1,00	0,08-0,89	0,04-1,00
	Стандартне відхилення	0,155	0,183	0,038	0,413	0,246	0,324
Coverage	Середнє	0,99		0,53		0,74	
	Діапазон	0,85-1,00		0,003-1,00		0,04-1,00	
	Стандартне відхилення	0,014		0,469		0,362	

[4]*Джерело: розраховано і сформовано автором*

Аналіз агрегованих метрик, наведених у таблиці, дозволяє кількісно підтвердити висновки, які візуально спостерігаються на рисунках 3.15 та 3.16. Насамперед простежується суттєва різниця у поведінці моделі залежно від домену навчання та тестування. У режимі in-domain, тобто за статистичної узгодженості між навчальним і тестовим середовищами, (рис. 3.15) модель демонструє стабільно високі результати: Accuracy ≈ 0.97 у номінальному та селективному режимах, F1 ≈ 0.89 та ≈ 0.87 відповідно, при невеликих значеннях стандартного відхилення ($\sigma \approx 0.05–0.18$ залежно від метрики). Це свідчить не лише про високу точність, а й про статистичну стабільність роботи системи в часі. Таким чином, у контрольованому середовищі дворівнева архітектура реалізує близький до ідеального сценарій функціонування.

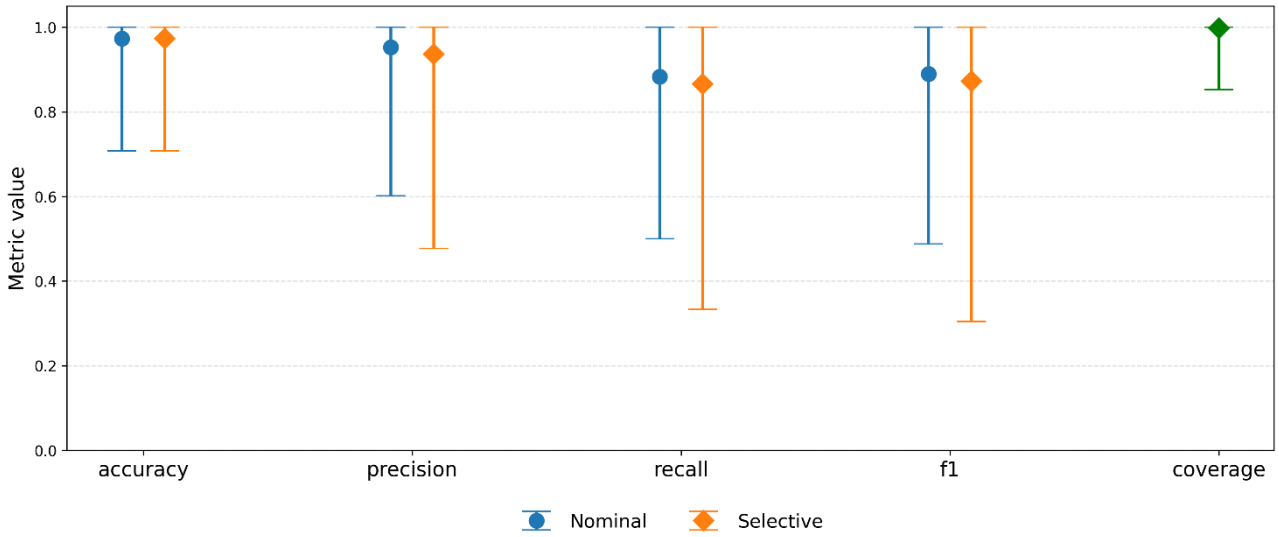

Рис. 3.15. Агреговані метрики ефективності дворівневої моделі прийняття рішень у номінальному та селективному режимах класифікації (in-domain)

(*Джерело: сформовано автором)*

Ситуація принципово змінюється у cross-domain (рис. 3.16) режимі для моделі, навченої на USB-IDS-1. У номінальному режимі середнє значення Accuracy знижується до ≈ 0.04, а F1 – до ≈ 0.04, що фактично означає втрату узагальнювальної здатності при перенесенні на інший статистичний профіль трафіку. Важливо підкреслити, що стандартні відхилення для цих показників залишаються відносно невеликими (σ ≈ 0.054 для Accuracy та σ ≈ 0.038 для F1), що вказує не на випадкові флуктуації, а на системний характер деградації. Модель стабільно працює некоректно в новому домені, що підтверджує чутливість алгоритмів до зміни розподілів ознак.

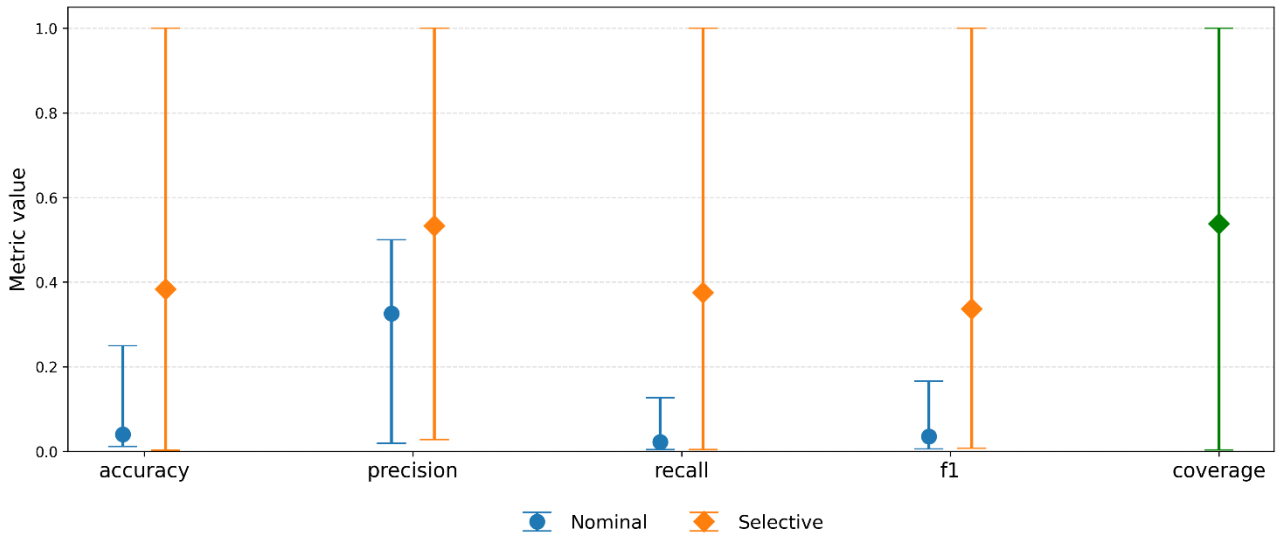

Рис. 3.16. Агреговані метрики ефективності дворівневої моделі прийняття рішень у номінальному та селективному режимах класифікації (cross-domain)

(*Джерело: сформовано автором)*

Перехід до селективного режиму у cross-domain умовах частково компенсує втрату якості: Accuracy зростає до ≈ 0.38, F1 – до ≈ 0.33, precision – до ≈ 0.53, recall – до ≈ 0.37. Хоча ці значення залишаються суттєво нижчими за in-domain результати, спостерігається майже восьмикратне зростання F1 порівняно з номінальним режимом. Одночасно відбувається зменшення coverage, що означає підвищення селективності прийняття рішень. Отже, механізм відмови від рішення дозволяє зменшити кількість грубих помилок атрибуції ціною зменшення частки автоматичних класифікацій, що є прийнятним компромісом в умовах невизначеності.

У mixed-domain режимі (рис. 3.17) показники займають проміжне положення. У номінальному режимі Accuracy ≈ 0.49, F1 ≈ 0.41, а у селективному – відповідно ≈ 0.68 та ≈ 0.62. При цьому стандартні відхилення є відносно високими (σ ≈ 0.34–0.37 для Accuracy, σ ≈ 0.24–0.32 для F1), що свідчить про значну варіативність між часовими вікнами. Це означає, що модель частково переноситься на новий домен, однак її робота залишається нестабільною через неоднорідність статистичного профілю змішаного середовища. Селективний режим у цьому випадку виконує стабілізуючу функцію, підвищуючи середні значення метрик та зменшуючи частку помилкових рішень.

Отримані результати дозволяють сформулювати наступні висновки: по-перше, високі значення метрик, досягнуті для in-domain, слід інтерпретувати як верхню межу ефективності запропонованої архітектури за умов статистичної узгодженості, а не як гарантований показник роботи в довільному середовищі. По-друге, різке зниження якості у cross-domain режимі не є наслідком некоректності алгоритмів як таких, а відображає відмінності у розподілах ознак, балансі класів та умовах формування трафіку. По-третє, запровадження селективної політики прийняття рішень дозволяє зменшити негативний вплив доменного зсуву шляхом контролю ризику хибної атрибуції.

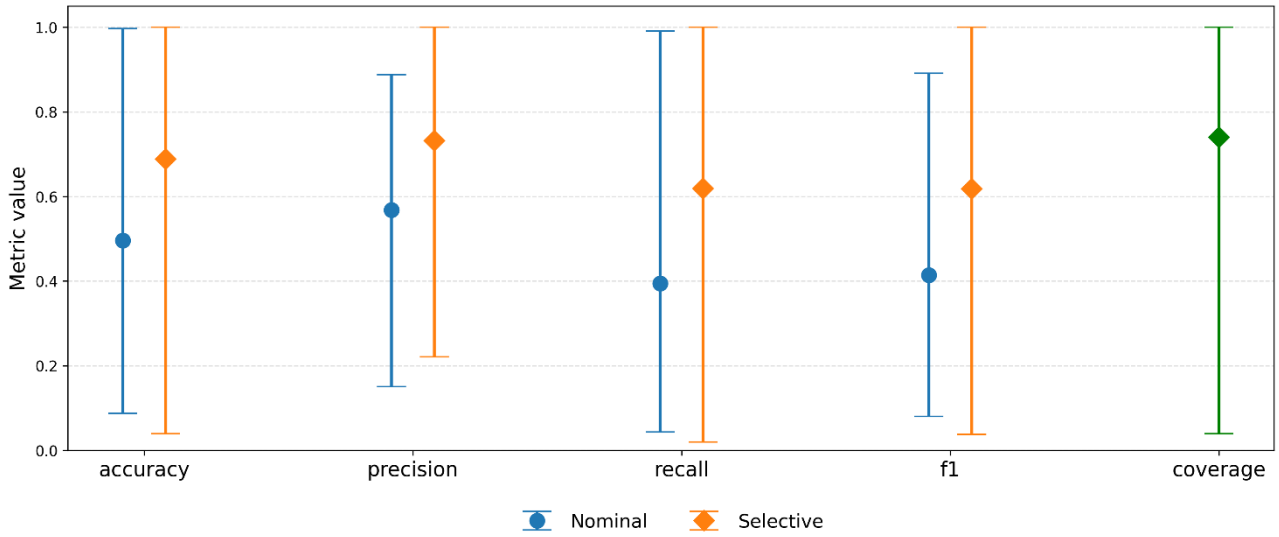

Рис. 3.17. Агреговані метрики ефективності дворівневої моделі прийняття рішень у номінальному та селективному режимах класифікації (mixed-domain)

(*Джерело: сформовано автором)*

Таким чином, аналіз таблиці підтверджує необхідність середовищно-орієнтованого навчання та адаптивних механізмів прийняття рішень у системах виявлення вторгнень, оскільки без урахування доменного контексту навіть формально коректні алгоритми машинного навчання втрачають здатність до узагальнення.

Аналіз матриць помилок

Агреговані матриці помилок дозволяють здійснити детальний аналіз структури класифікаційних рішень у режимі in-domain та доповнюють інтегральні метрики (Accuracy, Precision, Recall, F1), відображаючи характер і напрямок помилок. На відміну від узагальнених показників, матриця помилок демонструє, між якими саме класами виникає плутанина, що є критично важливим для оцінювання ефективності багатокласової системи виявлення вторгнень (рис. 3.18).

Аналіз абсолютної матриці помилок свідчить про домінування коректних класифікацій для більшості класів. Зокрема, для класів normative, Hulk та TCPFlood кількість правильних рішень суттєво перевищує кількість помилок. Це узгоджується з високими значеннями Accuracy (≈0.97) та F1 (≈0.89) у in-domain режимі. Клас TCPFlood демонструє практично повну відокремленість у просторі ознак, що вказує на чітку статистичну специфіку цієї атаки. Аналогічно, для Hulk

спостерігається мінімальна кількість перехресних помилок, що свідчить про добре сформовані межі рішень другого рівня класифікації.

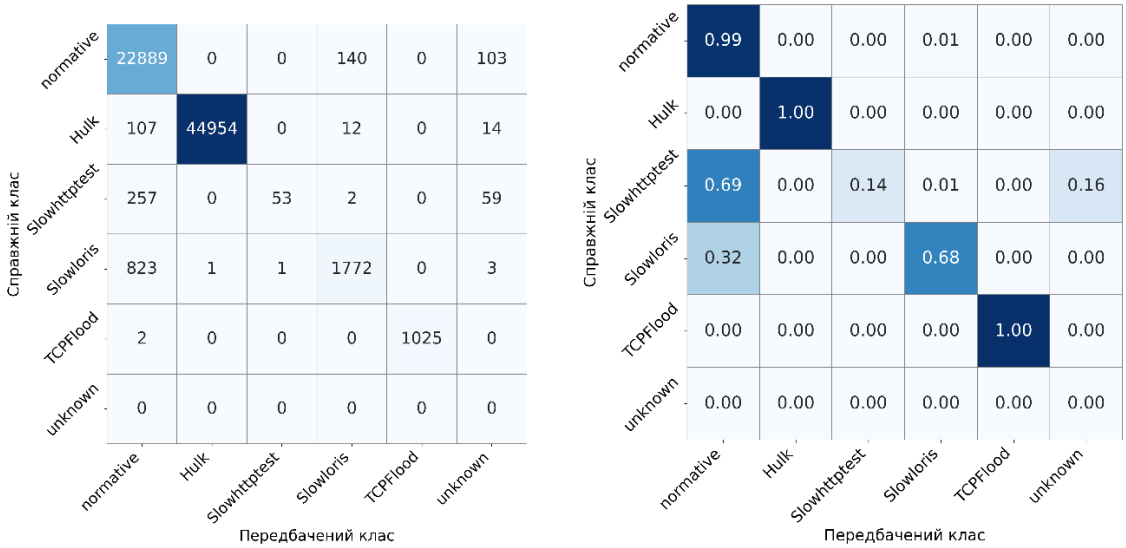

Рис. 3.18 – Матриці помилок дворівневої моделі класифікації (in-domain, зліва – абсолютні значення, справа – нормалізовані за рядками)

(*Джерело: сформовано автором)*

Водночас матриця виявляє локальні проблемні зони, які не очевидні з інтегральних метрик. Найбільша кількість помилок припадає на класи Slowhttptest та Slowloris. Нормалізована матриця показує, що значна частка об’єктів Slowhttptest (понад половину) помилково класифікується як нормативний трафік, а для Slowloris близько третини зразків також інтерпретується як normative. Це свідчить про часткове перекриття статистичних ознак повільних DoS-атак із легітимними повільними з’єднаннями. Таким чином, основна складність багатокласової атрибуції в межах узгодженого домену пов’язана не з плутаниною між різними типами атак, а з відмежуванням повільних атак від нормального трафіку.

Агреговані матриці помилок у cross-domain режимі відображають структурну зміну поведінки системи при перенесенні моделі в статистично відмінне середовище (рис. 3.19). На відміну від in-domain випадку, де більшість об’єктів коректно розподілялися між відповідними класами, у cross-domain спостерігається різке зростання частки рішень у стовпці unknown. Це означає, що система не здійснює примусової атрибуції типу атаки за відсутності достатньої впевненості, а переходить у режим відмови. Таким чином, значна частина зразків

класів Hulk та TCPFlood повністю переводиться у unknown, що свідчить про втрату узагальнювальної здатності багатокласового рівня в новому домені.

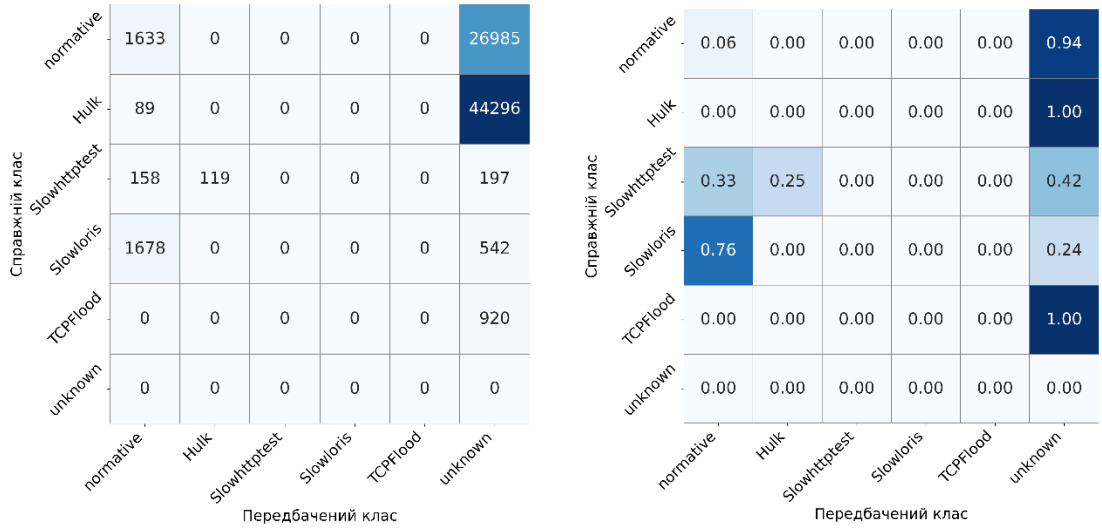

Рис. 3.19 – Матриці помилок дворівневої моделі класифікації (cross-domain, зліва – абсолютні значення, справа – нормалізовані за рядками)

(*Джерело: сформовано автором)*

Водночас для повільних атак (Slowloris, Slowhttptest) спостерігається часткове зміщення у бік класу normative, що вказує на перекриття статистичних ознак із легітимним трафіком. Проте ключовою характеристикою є саме зростання частки відмов, а не хаотичний перерозподіл між класами атак. Це означає, що механізм селективності виконує стабілізуючу функцію: при суттєвому доменному зсуві система обмежує автоматичну атрибуцію, знижуючи ризик хибних рішень. Отже, матриці помилок демонструють не лише деградацію якості при перенесенні, а й адаптивну реакцію архітектури через механізм контрольованої відмови.

Агреговані матриці помилок для mixed-domain режиму відображають проміжний характер узагальнювальної здатності моделі між in-domain та cross-domain сценаріями (рис 3.20).

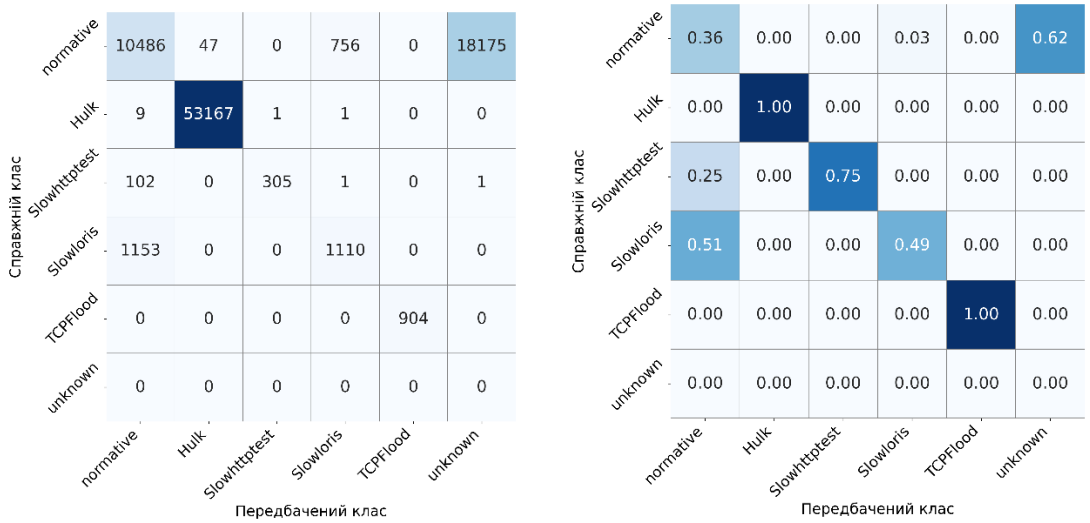

Рис. 3.20 – Матриці помилок дворівневої моделі класифікації (cross-domain, зліва – абсолютні значення, справа – нормалізовані за рядками)

(*Джерело: сформовано автором)*

У цьому випадку частина класів зберігає високу якість розпізнавання, тоді як інші демонструють помірну деградацію та часткове зміщення рішень. Зокрема, класи Hulk та TCPFlood практично повністю класифікуються коректно (нормалізовані значення близькі до 1.00), що свідчить про збереження характерних статистичних ознак цих атак навіть у змішаному середовищі. Натомість для класів Slowloris та Slowhttptest спостерігається помітна плутанина з класом normative: приблизно 51% зразків Slowloris та 25% Slowhttptest інтерпретуються як нормативний трафік. Це вказує на часткове перекриття статистичних характеристик повільних атак із легітимними потоками у змішаному середовищі.

Особливістю mixed-domain режиму є також значна частка відмов для класу normative (приблизно 62% переходить у unknown). Це означає, що система в умовах неоднорідного статистичного профілю частіше утримується від автоматичної атрибуції, коли рівень впевненості недостатній. Таким чином, механізм селективності виконує роль адаптивного обмежувача помилок: замість масової хибної класифікації модель переводить частину об’єктів у стан невизначеності. Загалом матриці помилок у mixed-domain режимі демонструють, що при частковому доменному зсуві зберігається здатність до коректної атрибуції

окремих типів атак, однак для статистично близьких до нормативного трафіку класів виникає перекриття, яке компенсується механізмом відмови.

Аналіз графіків тривалості обробки вікон (рис 3.21 та 3.22) та узагальненої таблиці 3.4 дозволяє виділити три ключові закономірності. По-перше, в обох режимах визначальним фактором сумарного часу інференсу є багатокласовий рівень. Середній час бінарної обробки практично не відрізняється між доменами (≈5.38 мс в in-domain та ≈5.65 мс у cross-domain), що підтверджує його стабільність і мінімальний внесок у загальну латентність.

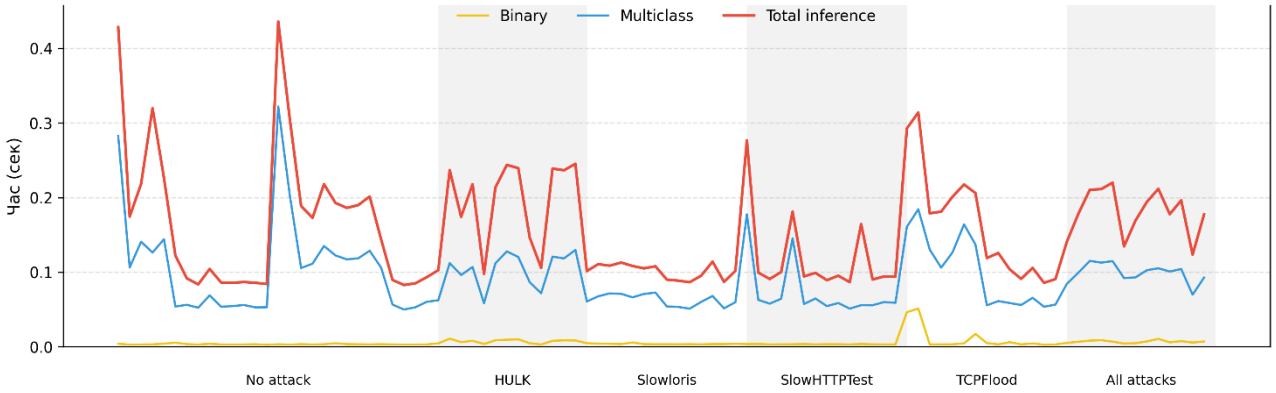

Рис. 3.21. Тривалість обробки 60-секундних вікон для моделі з набором USB-IDS-1 (*Джерело: сформовано автором)*

Натомість багатокласовий рівень демонструє суттєву різницю: у in-domain середній час становить ≈63.26 мс, тоді як у cross-domain він зростає до ≈93.00 мс. Саме цей компонент формує структуру сумарної кривої на графіках.

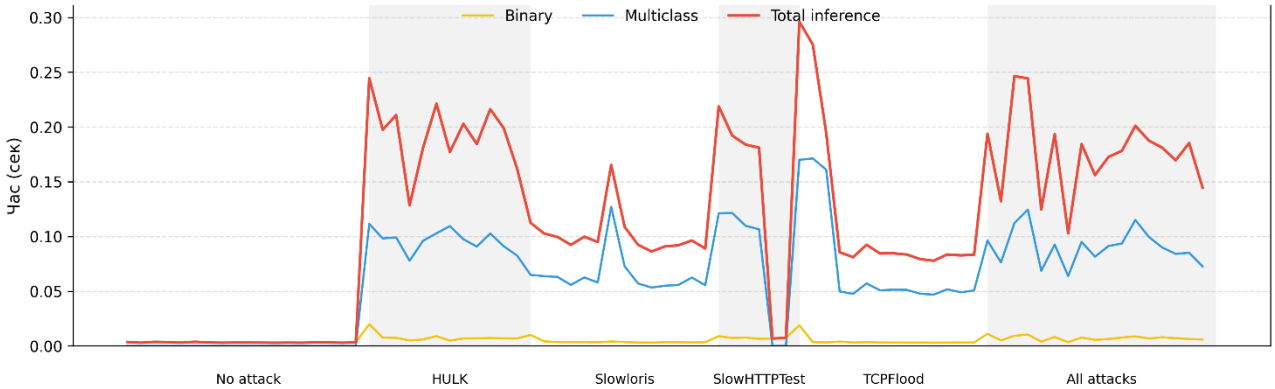

Рис. 3.22. Тривалість обробки 60-секундних вікон для моделі з набором власних даних (*Джерело: сформовано автором)*

Таблиця 3.4

Часові характеристики обробки 60-секундних вікон[5]

		Бінарна	Багатокласова	Загалом
Cross-domain	Середнє, мс	5.65	93	156.47
	Діапазон, мс	2.84-51.31	49.99-322.06	82.87-435.888
	Стандартне відхилення, мс	6.77	47.13	74.83
In-domain	Середнє, мс	5.38	63.26	114.55
	Діапазон, мс	2.94-19.85	0.0-171.28	2.90-296.24
	Стандартне відхилення, мс	3.13	44.91	81.29

[5]*Джерело: розраховано і сформовано автором*

По-друге, доменний зсув впливає не лише на якість класифікації, але й на обчислювальне навантаження. У cross-domain режимі спостерігається збільшення як середнього сумарного часу (≈156.47 мс проти ≈114.55 мс у in-domain), так і його варіативності (діапазон до ≈435.9 мс проти ≈296.2 мс). Це пояснюється зміною частки потоків, що передаються на другий рівень через менш узгоджену межу рішень першого класифікатора. Таким чином, статистична невідповідність між доменами призводить до збільшення кількості об’єктів, які потребують складнішої багатокласової обробки, що безпосередньо відображається на латентності.

По-третє, навіть за умов cross-domain режиму система зберігає обчислювальну ефективність. Максимальний зафіксований час обробки одного 60-секундного вікна не перевищує приблизно 0.44 с, що становить менше 1% від тривалості самого інтервалу. Отже, запропонована дворівнева архітектура є масштабованою та придатною до роботи в режимі, наближеному до реального часу. Графіки та таблиця разом підтверджують, що доменний зсув підвищує навантаження на багатокласовий рівень, але не створює критичних обмежень для практичної експлуатації системи.

Також було проаналізовано тривалість обробки моделлю одного потоку. Для цього час обробки кожного 60-секундного вікна було поділено на кількість потоків у ньому, а отримані значення усереднено та зведено в таблицю. Таким чином, наведені показники відображають середній час обробки одного потоку, незалежно від кількості трафіку у вікні (рис. 3.23 та 3.24).

Графіки демонструють, що бінарний рівень має мінімальний і практично незмінний внесок у латентність: ≈0.009–0.01 мс на потік у обох доменах. Основне навантаження припадає на багатокласовий рівень. У in-domain (рис. 3.23) режимі його середній час становить ≈2.762 мс (з піками до 121.492 мс), тоді як у cross-domain (рис. 3.24) – ≈0.22 мс (до 1.009 мс). Відповідно, сумарний середній час інференсу становить ≈4.482 мс у in-domain та ≈0.37 мс у cross-domain.

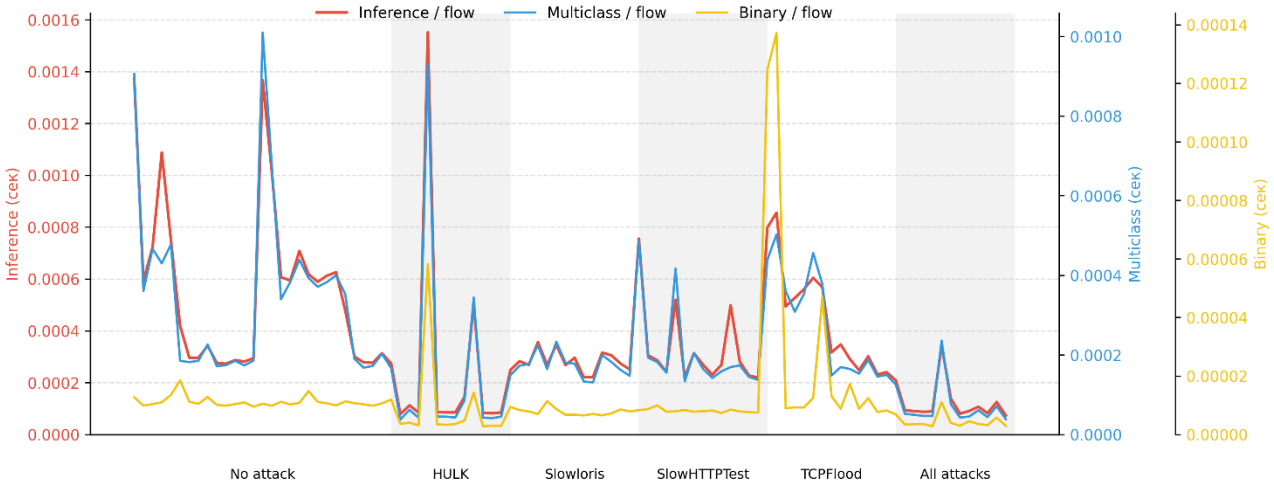

Рис. 3.23. Тривалість обробки мережевих потоків для моделі з набором USB-IDS-1 (*Джерело: сформовано автором)*

У таблиці 3.5 наведено час інференсу в перерахунку на один потік, що дозволяє оцінити швидкість обробки одного потоку складність незалежно від кількості трафіку у вікні. Бінарний рівень є стабільним у обох доменах: середній час становить ≈0.009–0.01 мс при мінімальній варіативності. Основне навантаження припадає на багатокласовий рівень: у cross-domain середній час становить ≈0.22 мс, тоді як у in-domain – ≈2.76 мс, що пов’язано з більшою часткою потоків, які проходять повну атрибуцію за відсутності механізму відмови.

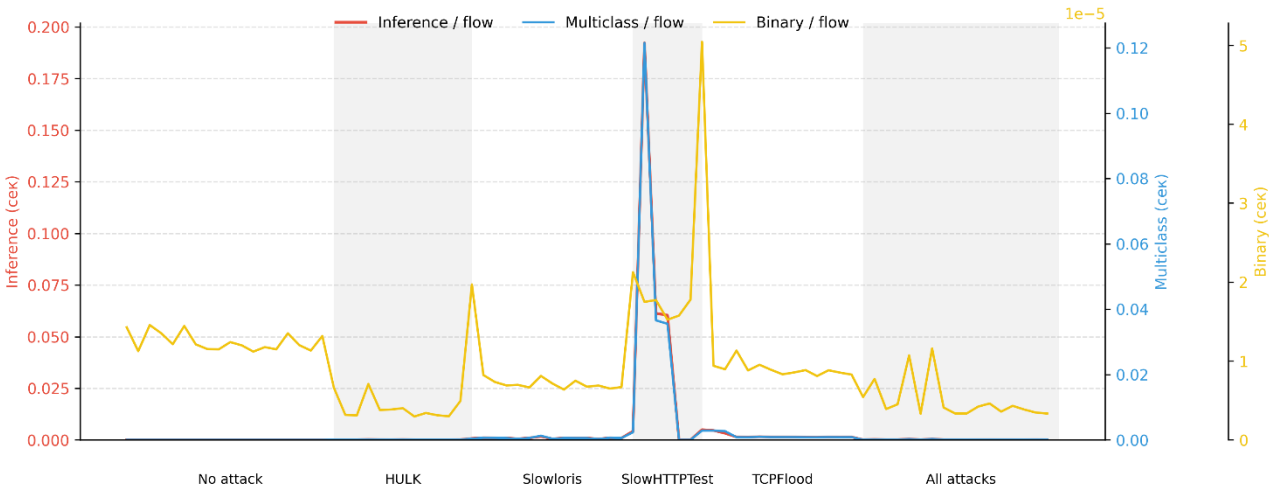

Рис. 3.24. Тривалість обробки мережевих потоків для моделі з набором власних даних (*Джерело: сформовано автором)*

Таблиця 3.5.

Часові характеристики обробки мережевих пакетів[6]

		Бінарна	Багатокласова	Загалом
Cross-domain	Середнє, мс	0.01	0.22	0.37
	Діапазон, мс	0.003-0.137	0.037-1.009	0.073-1.551
	Стандартне відхилення, мс	0.018	0.187	0.289
In-domain	Середнє, мс	0.009	2.762	4.482
	Діапазон, мс	0.002-0.050	0.000-121.492	0.011-192.252
	Стандартне відхилення, мс	0.006	14.48	23.14

[6]*Джерело: розраховано і сформовано автором*

Сумарний середній час інференсу на один потік становить ≈0.37 мс у cross-domain та ≈4.48 мс у in-domain. Навіть у максимальних випадках (≈1.55 мс та ≈192.25 мс відповідно) ці значення залишаються малими порівняно з тривалістю самого вікна (60 000 мс). Таким чином, обробка одного потоку займає менше ніж 0.01% тривалості вікна, а сумарна обробка всього трафіку у межах вікна становить менш ніж 1% його тривалості.

Окремої уваги заслуговує співвідношення між тривалістю аналізованого інтервалу (60 секунд, тобто 60 000 мс) та фактичним часом обробки одного вікна

трафіку. У in-domain режимі середній сумарний час інференсу для всього вікна становить ≈114.55 мс, а у пікових випадках може досягати ≈296 мс. У cross-domain режимі ці значення становлять відповідно ≈156.47 мс та ≈436 мс. Навіть максимальні зафіксовані значення становлять менше ніж 1% від тривалості самого 60-секундного інтервалу. У відносному вимірі це означає, що система витрачає на обробку вікна приблизно 0.2–0.7% часу його формування. Таким чином, запропонована архітектура працює з суттєвим часовим запасом і є придатною до функціонування в режимі, наближеному до реального часу, без створення критичної обчислювальної затримки.

Висновки до третього розділу

У цьому розділі узагальнено результати практичної реалізації запропонованої інформаційної технології та експериментальної перевірки дворівневої інтелектуальної системи виявлення мережевих атак у віртуальному середовищі. Реалізований програмний комплекс поєднує двоступеневу класифікацію (бінарний рівень на основі Naive Bayes та багатокласовий рівень на основі Random Forest), власний модуль захоплення та агрегації потоків, а також модульну архітектуру тренувальних компонентів. Така структура забезпечує повний цикл обробки даних – від перехоплення сирого трафіку до прийняття рішення з урахуванням порогової політики та можливості відмови (unknown). Емуляційне середовище, побудоване за допомогою GNS3 та VirtualBox, дозволило безпечно відтворити типові сценарії DDoS-атак (HULK, Slowloris, SlowHTTPTest, TCPFlood) разом із нормативним трафіком, що імітує звичайну мережеву активність, і сформувати детерміновано маркований набір потокових даних.

Експериментальний аналіз показав, що у статистично узгодженому середовищі (in-domain) система демонструє високі та стабільні показники якості (Accuracy ≈ 0.97, F1 ≈ 0.89) при незначній варіативності між часовими вікнами. Коливання інтегральної міри зсуву домену S_{domain} у межах одного середовища не призводять до суттєвої деградації метрик, що свідчить про робастність моделі до внутрішньодоменної мінливості трафіку. Натомість у cross-domain режимі

зафіксовано різке зниження номінальних метрик (F1 ≈ 0.04), що підтверджує чутливість моделей машинного навчання до зміни статистичного профілю ознак. Кількісний аналіз показав, що міждоменна розбіжність суттєво перевищує внутрішньодоменну, що пояснює втрату узагальнювальної здатності при перенесенні.

Важливим результатом є підтвердження ефективності селективного механізму прийняття рішень. У режимах cross-domain та mixed-domain застосування порогової політики та класу «unknown» дозволило частково компенсувати деградацію якості: підвищити Precision та F1 за рахунок зниження coverage і обмеження хибної багатокласової атрибуції. Агреговані матриці помилок показали, що у разі суттєвого доменного зсуву система переходить до обережної стратегії, збільшуючи частку відмов замість генерування помилкових рішень. У mixed-domain режимі продемонстровано проміжну поведінку: часткове збереження узагальнення для окремих класів атак та підвищену варіативність метрик, що стабілізується завдяки селективності.

Дослідження часових характеристик підтвердило обчислювальну ефективність архітектури. Середній час обробки одного 60-секундного вікна становить близько 115–156 мс залежно від домену, що не перевищує 1% тривалості самого інтервалу. У перерахунку на один потік середній час інференсу становить ≈4.48 мс у in-domain та ≈0.37 мс у cross-domain режимі, що свідчить про лінійну масштабованість алгоритму та придатність системи до роботи в режимі, наближеному до реального часу. Модульність реалізації забезпечує узгодженість між етапами збору, навчання та оцінювання моделей, а також можливість подальшого розширення функціональності.

Отже, результати розділу підтверджують працездатність і відтворюваність запропонованої інформаційної технології, демонструють її стабільність у межах узгодженого домену та виявляють критичну залежність якості класифікації від статистичного зсуву розподілів ознак. Це обґрунтовує доцільність інтеграції адаптивних порогових механізмів у системах виявлення вторгнень та визначає напрям подальших досліджень, спрямованих на підвищення переносимості моделей у гетерогенних мережевих середовищах.

ВИСНОВКИ

У дисертаційній роботі вирішено актуальне науково-практичне завдання щодо створення нової інформаційної технології дворівневої інтелектуальної системи аналізу мережевих атак. Основні наукові та практичні результати полягають у наступному.

1. Проведено аналіз релевантних методів машинного навчання в системах розпізнавання вторгнень (IDS), який підтвердив, що головним обмеженням чинних рішень є деградація точності в умовах зсуву домену (domain shift). Обґрунтовано доцільність застосування дворівневої ієрархічної класифікації, що дозволить відокремити етап швидкої фільтрації від поглибленої класифікації атак.

2. Удосконалено метод виявлення вторгнень шляхом синтезу моделі налаштування порогів класифікації. На відміну від наявних аналогів, запропонована модель базується на критерії мінімізації баєсівського ризику та використовує дивергенцію Кульбака-Лейблера для кількісної оцінки статистичних розбіжностей трафіку, що забезпечує автоматичне коригування чутливості системи до змін мережевого середовища.

3. Розроблено архітектуру інформаційної технології аналізу мережевих атак, яка вперше реалізує замкнений цикл «генерація–маркування–класифікація». Доведено, що впровадження механізму відмови від рішення та спеціального класу невизначеності «Unknown» дозволило підвищити вірогідність розпізнавання атак в умовах неповної інформації та зменшити обчислювальне навантаження на систему розпізнавання вторгнень.

4. Створено комплекс інструментальних засобів модуль FlowCapture та ін. та ізольоване віртуальне середовище на базі GNS3 для безпечної емуляції складних сценаріїв атак, зокрема DDoS. Експериментально підтверджено ефективність розробленої системи. На верифікованих наборах даних досягнуто точності класифікації $Accuracy \approx 0.97$ та $F1 \approx 0.7$, що свідчить про адекватність розроблених моделей умовам варіативності мережевого трафіку.

5. Набув подальшого розвитку метод оцінювання ефективності систем виявлення та класифікації мережевих атак. Запропонований перехід від

статичних метрик до синхронного віконного аналізу показників надійності дозволив здійснювати моніторинг якості роботи IDS у синхронному режимі та своєчасно виявляти момент втрати релевантності навчених моделей класифікації вторгнень.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Штанько В., Нікітенко Є. Проєктування та реалізація віртуального середовища для аналізу мережевого трафіку. *Наука і техніка сьогодні*. 7(48). DOI:10.52058/2786-6025-2025-7(48)-2028-2045.
2. A. Alissa K., S. Alrayes F., Tarmissi K. та ін. Planet Optimization with Deep Convolutional Neural Network for Lightweight Intrusion Detection in Resource-Constrained IoT Networks. *Applied Sciences*. Вип. 12, № 17. С. 8676. DOI:10.3390/app12178676.
3. Ackermann T., Karch M., Kippe J. Integration of Cyber Threat Intelligence into Security Onion and Malcolm for the use case of industrial networks. *at - Automatisierungstechnik*. Вип. 71, № 9. С. 802–815. DOI:10.1515/auto-2023-0057.
4. Ahmed Hamza M., Hassan Abdalla Hashim A., G. Mohamed H. та ін. Hyperparameter Tuned Deep Learning Enabled Intrusion Detection on Internet of Everything Environment. *Computers, Materials & Continua*. Вип. 73, № 3. С. 6579–6594. DOI:10.32604/cmc.2022.031303.
5. Alissa K. A., Alotaibi S. S., Alrayes F. S. та ін. Crystal Structure Optimization with Deep-Autoencoder-Based Intrusion Detection for Secure Internet of Drones Environment. *Drones*. Вип. 6, № 10. С. 297. DOI:10.3390/drones6100297.
6. Asad H., Gashi I. Dynamical analysis of diversity in rule-based open source network intrusion detection systems. *Empirical Software Engineering*. Вип. 27, № 1. С. 4. DOI:10.1007/s10664-021-10046-w.
7. Ashraf J., Keshk M., Moustafa N. та ін. IoTBoT-IDS: A novel statistical learning-enabled botnet detection framework for protecting networks of smart cities. *Sustainable Cities and Society*. Вип. 72, 09.2021. С. 103041. DOI:10.1016/j.scs.2021.103041.
8. Barros W. K. P., Barbosa M. T., Dias L. A. та ін. Fully Parallel Proposal of Naive Bayes on FPGA. *Electronics*. Вип. 11, № 16. С. 2565. DOI:10.3390/electronics11162565.
9. Boyko N., Omeliukh R., Duliaba N. The Random Forest Algorithm as an Element of Statistical Learning for Disease Prediction.
10. Breiman L. Random Forests. *Machine Learning*. Вип. 45, № 1. С. 5–32. DOI:10.1023/A:1010933404324.
11. Cabrera J. B. D., Gutiérrez C., Mehra R. K. Ensemble methods for anomaly detection and distributed intrusion detection in Mobile Ad-Hoc Networks. *Information Fusion*. Вип. 9, № 1. С. 96–119. DOI:10.1016/j.inffus.2007.03.001.
12. Chamkar S. A., Zaydi M., Maleh Y. та ін. Improving Threat Detection in Wazuh Using Machine Learning Techniques. *Journal of Cybersecurity and Privacy*. Вип. 5, № 2. С. 34. DOI:10.3390/jcp5020034.
13. Chawla A. Out-of-core learning with sklearn. A Lesser-Known Feature of Sklearn To Train Models on Large Datasets. Out-of-core learning with sklearn. 20.04.2023. URL: https://blog.dailydoseofds.com/p/a-lesser-known-feature-of-sklearn (дата звернення: 19.10.2025).
14. DataDriven. Огляд ринку кібербезпеки в Україні. (01.2025). URL: https://itukraine.org.ua/files/Ukraine-Cybersec-Market-Review.pdf 2025.

15. De Nascimento C. M., Hou J. Uncertainty-Aware Adaptive Intrusion Detection Using Hybrid CNN-LSTM with cWGAN-GP Augmentation and Human-in-the-Loop Feedback. *Safety*. Вип. 11, № 4. С. 120. DOI:10.3390/safety11040120.
16. Depren O., Topallar M., Anarim E. та ін. An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. *Expert Systems with Applications*. Вип. 29, № 4. С. 713–722. DOI:10.1016/j.eswa.2005.05.002.
17. D'hooge L., Verkerken M., Wauters T. та ін. Investigating Generalized Performance of Data-Constrained Supervised Machine Learning Models on Novel, Related Samples in Intrusion Detection. *Sensors*. Вип. 23, № 4. С. 1846. DOI:10.3390/s23041846.
18. Domingos P., Pazzani M. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*. Вип. 29, № 2–3. С. 103–130. DOI:10.1023/A:1007413511361.
19. Duda R. O., Hart P. E., Stork D. G. Pattern classification. 2. ed. New York : Wiley, 2001. 654 с. ISBN 978-0-471-05669-0.
20. Duhayyim M. A., Alissa K. A., Alrayes F. S. та ін. Evolutionary-Based Deep Stacked Autoencoder for Intrusion Detection in a Cloud-Based Cyber-Physical System. *Applied Sciences*. Вип. 12, № 14. С. 6875. DOI:10.3390/app12146875.
21. Elkan C. The Foundations of Cost-Sensitive Learning. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligenc*. № 1.
22. Fatani A., Dahou A., Al-qaness M. A. A. та ін. Advanced Feature Extraction and Selection Approach Using Deep Learning and Aquila Optimizer for IoT Intrusion Detection System. *Sensors*. Вип. 22, № 1. С. 140. DOI:10.3390/s22010140.
23. G. Mohamed H., Alrowais F., Abdullah Al-Hagery M. та ін. Optimal Wavelet Neural Network-Based Intrusion Detection in Internet of Things Environment. *Computers, Materials & Continua*. Вип. 75, № 2. С. 4467–4483. DOI:10.32604/cmc.2023.036822.
24. G. Mohamed H., S. Alotaibi S., M. Eltahir M. та ін. Feature Selection with Stacked Autoencoder Based Intrusion Detection in Drones Environment. *Computers, Materials & Continua*. Вип. 73, № 3. С. 5441–5458. DOI:10.32604/cmc.2022.031887.
25. Gama J., Žliobaitė I., Bifet A. та ін. A survey on concept drift adaptation. *ACM Computing Surveys*. Вип. 46, № 4. С. 1–37. DOI:10.1145/2523813.
26. Geurts P., Ernst D., Wehenkel L. Extremely randomized trees. *Machine Learning*. Вип. 63, № 1. С. 3–42. DOI:10.1007/s10994-006-6226-1.
27. Gogoi P., Bhattacharyya D. K., Borah B. та ін. MLH-IDS: A Multi-Level Hybrid Intrusion Detection Method. *The Computer Journal*. Вип. 57, № 4. С. 602–623. DOI:10.1093/comjnl/bxt044.
28. Gold S. The future of the firewall. *Network Security*. Вип. 2011, № 2. С. 13–15. DOI:10.1016/S1353-4858(11)70015-0.
29. Gopalan S. S. Towards Effective Detection of Botnet Attacks using BoT-IoT Dataset. 2021.
30. Heino J., Hakkala A., Virtanen S. Study of methods for endpoint aware inspection in a next generation firewall. *Cybersecurity*. Вип. 5, № 1. С. 25. DOI:10.1186/s42400-022-00127-8.
31. Hoeve M. Detecting intrusions in encrypted control traffic. *Proceedings of the first ACM workshop on Smart energy grid securityCCS'13: 2013 ACM SIGSAC Conference*

on Computer and Communications Security. (08.11.2013). Berlin Germany : ACM, 2013. DOI:10.1145/2516930.2516945. С. 23–28.
32. Huang H., Zhang H. An Online Intrusion Detection Method using Adaptive Multi-Level Classifier Network and PCA-Guided Model Reuse Mechanism. *Proceedings of the 2025 4th International Conference on Cryptography, Network Security and Communication TechnologyCNSCT 2025: 2025 4th International Conference on Cryptography, Network Security and Communication Technology*. (17.01.2025). Zhengzhou China : ACM, 2025. DOI:10.1145/3723890.3723894. С. 16–20.
33. Improved Intrusion Detection Algorithm based on TLBO and GA Algorithms. *The International Arab Journal of Information Technology*. Вип. 18, № 2. DOI:10.34028/iajit/18/2/5.
34. Issa M. M., Aljanabi M., Muhialdeen H. M. Systematic literature review on intrusion detection systems: Research trends, algorithms, methods, datasets, and limitations. *Journal of Intelligent Systems*. Вип. 33, № 1. С. 20230248. DOI:10.1515/jisys-2023-0248.
35. Khraisat A., Gondal I., Vamplew P. та ін. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*. Вип. 2, № 1. С. 20. DOI:10.1186/s42400-019-0038-7.
36. Kifer D., Ben-David S., Gehrke J. Detecting Change in Data Streams. *Proceedings 2004 VLDB Conference*. Elsevier, 2004. С. 180–191. DOI:10.1016/B978-012088469-8.50019-X.
37. Koc L., Mazzuchi T. A., Sarkani S. A network intrusion detection system based on a Hidden Naïve Bayes multiclass classifier. *Expert Systems with Applications*. Вип. 39, № 18. С. 13492–13500. DOI:10.1016/j.eswa.2012.07.009.
38. Konyrbaev N., Nikitenko Y., Shtanko V. та ін. Evaluation and optimization of the naive bayes algorithm for intrusion detection systems using the USB-IDS-1 dataset. *Eastern-European Journal of Enterprise Technologies*. Вип. 6, 2 (132). С. 74–82. DOI:10.15587/1729-4061.2024.317471.
39. Kullback S., Leibler R. A. On Information and Sufficiency. *The Annals of Mathematical Statistics*. Вип. 22, № 1. С. 79–86. DOI:10.1214/aoms/1177729694.
40. Kunhare N., Tiwari R., Dhar J. Particle swarm optimization and feature selection for intrusion detection system. *Sādhanā*. Вип. 45, № 1. С. 109. DOI:10.1007/s12046-020-1308-5.
41. LabelEncoder. *scikit-learn*. URL: https://scikit-learn/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html (дата звернення: 18.10.2025).
42. Lakhno V., Lakhno M., Kryvoruchko O. та ін. Automation of DDoS attack investigation in industrial control systems using Bayesian networks on Python ⋆.
43. Liao H.-J., Richard Lin C.-H., Lin Y.-C. та ін. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*. Вип. 36, № 1. С. 16–24. DOI:10.1016/j.jnca.2012.09.004.
44. [Literature Review] Hierarchical Classification for Intrusion Detection System: Effective Design and Empirical Analysis. URL: https://www.themoonlight.io/en/review/hierarchical-classification-for-intrusion-detection-system-effective-design-and-empirical-analysis (дата звернення: 17.10.2025).

45. Liu M., Xue Z., Xu X. та ін. Host-Based Intrusion Detection System with System Calls: Review and Future Trends. *ACM Computing Surveys*. Вип. 51, № 5. С. 1–36. DOI:10.1145/3214304.
46. Mienye I. D., Swart T. G. Deep Autoencoder Neural Networks: A Comprehensive Review and New Perspectives. *Archives of Computational Methods in Engineering*. 15.03.2025. DOI:10.1007/s11831-025-10260-5.
47. MLfast.co R. | Precision, Recall and F1 Explained [With 10 ML Use case]. *Medium*. 15.04.2024. URL: https://rumn.medium.com/precision-recall-and-f1-explained-with-10-ml-use-case-6ef2fbe458e5 (дата звернення: 19.10.2025).
48. Mocean L., Vlad M.-P. Artificial Intelligence in Cybersecurity: Applications and Challenges. *Scientific Bulletin (Scientific Bulletin of the Nicolae Balcescu Land Forces Academy, Siblu, Romania)*. Вип. 30, № 2. С. 199–208. DOI:10.2478/bsaft-2025-0021.
49. Naive Bayes. *scikit-learn*. URL: https://scikit-learn/stable/modules/naive_bayes.html (дата звернення: 17.10.2025).
50. Pandelu A. P. Day 78: Modular Coding for Model Training – Building Smarter ML Pipelines. *Medium*. 13.01.2025. URL: https://medium.com/@bhatadithya54764118/day-78-modular-coding-for-model-training-building-smarter-ml-pipelines-6026e036c724 (дата звернення: 19.10.2025).
51. Park W., Ahn S. Performance Comparison and Detection Analysis in Snort and Suricata Environment. *Wireless Personal Communications*. Вип. 94, № 2. С. 241–252. DOI:10.1007/s11277-016-3209-9.
52. Paxson V. Bro: a system for detecting network intruders in real-time. *Computer Networks*. Вип. 31, № 23–24. С. 2435–2463. DOI:10.1016/S1389-1286(99)00112-7.
53. Prabhakaran V., Kulandasamy A. Hybrid semantic deep learning architecture and optimal advanced encryption standard key management scheme for secure cloud storage and intrusion detection. *Neural Computing and Applications*. Вип. 33, № 21. С. 14459–14479. DOI:10.1007/s00521-021-06085-5.
54. Qaddoura R., M. Al-Zoubi A., Faris H. та ін. A Multi-Layer Classification Approach for Intrusion Detection in IoT Networks Based on Deep Learning. *Sensors*. Вип. 21, № 9. С. 2987. DOI:10.3390/s21092987.
55. Ribeiro M. T., Singh S., Guestrin C. «Why Should I Trust You?»: Explaining the Predictions of Any Classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data MiningKDD '16: The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. (13.08.2016). San Francisco California USA : ACM, 2016. DOI:10.1145/2939672.2939778. С. 1135–1144.
56. Salau A. O., Beyene M. M. Software defined networking based network traffic classification using machine learning techniques. *Scientific Reports*. Вип. 14, № 1. С. 20060. DOI:10.1038/s41598-024-70983-6.
57. Sarnovsky M., Paralic J. Hierarchical Intrusion Detection Using Machine Learning and Knowledge Model. *Symmetry*. Вип. 12, № 2. С. 203. DOI:10.3390/sym12020203.
58. Sculley D., Holt G., Golovin D. та ін. Hidden Technical Debt in Machine Learning Systems. *Advances in Neural Information Processing Systems*. Вип. 28, 2015.
59. Shiravi A., Shiravi H., Tavallaee M. та ін. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*. Вип. 31, № 3. С. 357–374. DOI:10.1016/j.cose.2011.12.012.

60. Song W., Beshley M., Przystupa K. та ін. A Software Deep Packet Inspection System for Network Traffic Analysis and Anomaly Detection. *Sensors*. Вип. 20, № 6. С. 1637. DOI:10.3390/s20061637.
61. Teixeira D., Assunção L., Pereira T. та ін. OSSEC IDS Extension to Improve Log Analysis and Override False Positive or Negative Detections. *Journal of Sensor and Actuator Networks*. Вип. 8, № 3. С. 46. DOI:10.3390/jsan8030046.
62. Tomar D., Agarwal S. Twin Support Vector Machine: A review from 2007 to 2014. *Egyptian Informatics Journal*. Вип. 16, № 1. С. 55–69. DOI:10.1016/j.eij.2014.12.003.
63. Tudela K. N. B., Patilla H. J. Security Onion as a Network Auditing Tool at the San Cristóbal de Huamanga National University. *International Journal of Advanced Computer Science and Applications*. Вип. 16, № 3. DOI:10.14569/IJACSA.2025.0160314.
64. Tuning the hyper-parameters of an estimator. *scikit-learn*. URL: https://scikit-learn/stable/modules/grid_search.html (дата звернення: 19.10.2025).
65. Turner-Trauring I. Reducing Pandas memory usage #3: Reading in chunks. *Python⇒Speed*. 11.02.2020. URL: https://pythonspeed.com/articles/chunking-pandas/ (дата звернення: 18.10.2025).
66. VarianceThreshold. *scikit-learn*. URL: https://scikit-learn/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html (дата звернення: 18.10.2025).
67. Whig P., Kouser S., Bhatia A. B. та ін. Prediction of breast cancer diagnosis using random forest classifier. *Computational Intelligence and Modelling Techniques for Disease Detection in Mammogram Images*. Elsevier, 2024. С. 55–73. DOI:10.1016/B978-0-443-13999-4.00011-0.
68. Wickramasinghe I., Kalutarage H. Naive Bayes: applications, variations and vulnerabilities: a review of literature with code snippets for implementation. *Soft Computing*. Вип. 25, № 3. С. 2277–2293. DOI:10.1007/s00500-020-05297-6.
69. Wizards D. S. Machine learning pipeline: What it is, Why it matters, and Guide to Building it? *Medium*. 23.05.2023. URL: https://medium.com/@datasciencewizards/machine-learning-pipeline-what-it-is-why-it-matters-and-guide-to-building-it-2940d143fd37 (дата звернення: 19.10.2025).
70. Xu L., Mogos G. Bugs in Security Onion. *2021 6th International Conference on Systems, Control and Communications (ICSCC)ICSCC 2021: 2021 6th International Conference on Systems, Control and Communications*. (15.10.2021). Chongqing China : ACM, 2021. DOI:10.1145/3510362.3510363. С. 1–6.
71. Zadrozny B., Elkan C. Learning and making decisions when costs and probabilities are both unknown. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*(26.08.2001). San Francisco California : ACM, 2001. DOI:10.1145/502512.502540. С. 204–213.
72. Zoghi Z., Serpen G. Building an intrusion detection system on UNSW - NB15 : Reducing the margin of error to deal with data overlap and imbalance. *Concurrency and Computation: Practice and Experience*. Вип. 36, № 25. С. e8242. DOI:10.1002/cpe.8242.

73. Zybin S., Khoroshko V., Khokhlachova Y. та ін. Approach of the Attack Analysis to Reduce Omissions in the Risk Management. *CEUR Workshop Proceedings*. 2021. URL: https://ceur-ws.org/Vol-2923/paper35.pdf
74. Анна Корченко Методи ідентифікації аномальних станів для систем виявлення вторгнень. Монографія. Київ : ЦП «Компринт», 2019. 361 p. ISBN 978-966-929-874-4.
75. Бекетова Г. С., Ахметов Б. Б., Корченко О. Г. et al. Design of a model for intellectual detection of cyber-attacks, based on the logical procedures and the coverage matrices of features. *Ukrainian Scientific Journal of Information Security*. Vol. 22, Issue 3. P. 242–254. DOI:10.18372/2225-5036.22.11096.
76. Ahamed Md. K., Karim A. Cascaded intrusion detection system using machine learning. *Systems and Soft Computing*. Vol. 7, 01.12.2025. P. 200182. DOI:10.1016/j.sasc.2024.200182.
77. Albin E., Rowe N. C. A Realistic Experimental Comparison of the Suricata and Snort Intrusion-Detection Systems. *2012 26th International Conference on Advanced Information Networking and Applications Workshops2012 IEEE Workshops of International Conference on Advanced Information Networking and Applications (WAINA)*. (03.2012). Fukuoka, Japan : IEEE, 2012. DOI:10.1109/WAINA.2012.29. P. 122–127.
78. Alin F., Chemchem A., Nolot F. et al. Towards a Hierarchical Deep Learning Approach for Intrusion Detection. 2020. P. 15–27. DOI:10.1007/978-3-030-45778-5_2.
79. Amal Al-Harbi, Randa Jabeur. An Efficient Method for Detection of DDoS Attacks on the Web Using Deep Learning Algorithms. *International Journal of Advanced Trends in Computer Science and Engineering*. Vol. 10, Issue 4. P. 2821–2829. DOI:10.30534/ijatcse/2021/271042021.
80. Baktashmotlagh M., Harandi M. T., Lovell B. C. et al. Unsupervised Domain Adaptation by Domain Invariant Projection. *2013 IEEE International Conference on Computer Vision2013 IEEE International Conference on Computer Vision (ICCV)*. (12.2013). Sydney, Australia : IEEE, 2013. DOI:10.1109/ICCV.2013.100. P. 769–776.
81. Bishop C. M. Pattern recognition and machine learning. New York : Springer, 2006. 738 p. [Q327 .B52 2006]. ("Information science and statistics" Series). ISBN 978-0-387-31073-2.
82. Bondarovets S., Koval O., Hnatiuk S. Anomaly detection system for mobile carrier based on Big Data concept. *Collection «Information technology and security»*. Vol. 4, Issue 1. P. 44–53. DOI:10.20535/2411-1031.2016.4.1.96016.
83. Catillo M., Del Vecchio A., Ocone L. et al. USB-IDS-1: a Public Multilayer Dataset of Labeled Network Flows for IDS Evaluation. *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. (06.2021). Taipei, Taiwan : IEEE, 2021. DOI:10.1109/DSN-W52860.2021.00012. P. 1–6.
84. Chow C. On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*. Vol. 16, Issue 1. P. 41–46. DOI:10.1109/TIT.1970.1054406.
85. Creech G., Hu J. A Semantic Approach to Host-Based Intrusion Detection Systems Using Contiguousand Discontiguous System Call Patterns. *IEEE Transactions on Computers*. Vol. 63, Issue 4. P. 807–819. DOI:10.1109/TC.2013.13.

86. Farhat S., Abdelkader M., Meddeb-Makhlouf A. et al. Evaluation of DoS/DDoS Attack Detection with ML Techniques on CIC-IDS2017 Dataset: *Proceedings of the 9th International Conference on Information Systems Security and Privacy9th International Conference on Information Systems Security and Privacy*. (2023). Lisbon, Portugal : SCITEPRESS - Science and Technology Publications, 2023. DOI:10.5220/0011605700003405. P. 287–295.
87. Geng C., Huang S., Chen S. Recent Advances in Open Set Recognition: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 43, Issue 10. P. 3614–3631. DOI:10.1109/TPAMI.2020.2981604.
88. Haibo He, Garcia E. A. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*. Vol. 21, Issue 9. P. 1263–1284. DOI:10.1109/TKDE.2008.239.
89. Havrylova A., Korol O., Voropay N. et al. Analysis of cryptographic authentication and manipulation detection methods for big data. *Modern Information Security*. Vol. 57, Issue 1. DOI:10.31673/2409-7292.2024.010011.
90. Jadhav A. D., Pellakuri V. Highly accurate and efficient two phase-intrusion detection system (TP-IDS) using distributed processing of HADOOP and machine learning techniques. *Journal of Big Data*. Vol. 8, Issue 1. P. 131. DOI:10.1186/s40537-021-00521-y.
91. Júnior P. R. M., Boult T. E., Wainer J. et al. Open-Set Support Vector Machines. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*. Vol. 52, Issue 6. P. 3785–3798. DOI:10.1109/TSMC.2021.3074496.
92. Khamphakdee N., Benjamas N., Saiyod S. Improving Intrusion Detection System Based on Snort Rules for Network Probe Attacks Detection with Association Rules Technique of Data Mining. *Journal of ICT Research and Applications*. Vol. 8, Issue 3. P. 234–250. DOI:10.5614/itbj.ict.res.appl.2015.8.3.4.
93. Kruegel C., Mutz D., Robertson W. et al. Bayesian event classification for intrusion detection. *19th Annual Computer Security Applications Conference, 2003. Proceedings.19th Annual Computer Security Applications Conference, 2003.* (2003). Las Vegas, Nevada, USA : IEEE, 2003. DOI:10.1109/CSAC.2003.1254306. P. 14–23.
94. Lakhno V. Creation of the adaptive cyber threat detection system on the basis of fuzzy feature clustering. *Eastern-European Journal of Enterprise Technologies*. Vol. 2, Issue 9(80). P. 18. DOI:10.15587/1729-4061.2016.66015.
95. Loutskii H., Volokyta A., Yakushev O. et al. Development of real time method of detecting attacks based on artificial intelligence. *Technology audit and production reserves*. Vol. 3, Issue 1(29). P. 40. DOI:10.15587/2312-8372.2016.71677.
96. Mantovani R. G., Horváth T., Rossi A. L. D. et al. Better Trees: An empirical study on hyperparameter tuning of classification decision tree induction algorithms. *Data Mining and Knowledge Discovery*. Vol. 38, Issue 3. P. 1364–1416. DOI:10.1007/s10618-024-01002-5.
97. Moreno-Torres J. G., Saez J. A., Herrera F. Study on the Impact of Partition-Induced Dataset Shift on k-Fold Cross-Validation. *IEEE Transactions on Neural Networks and Learning Systems*. Vol. 23, Issue 8. P. 1304–1312. DOI:10.1109/TNNLS.2012.2199516.
98. Moustafa N., Slay J. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). *2015 Military Communications*

and Information Systems Conference (MilCIS)2015 Military Communications and Information Systems Conference (MilCIS). (11.2015). Canberra, Australia : IEEE, 2015. DOI:10.1109/MilCIS.2015.7348942. P. 1–6.
99. Neupane K., Haddad R., Chen L. Next Generation Firewall for Network Security: A Survey. *SoutheastCon 2018SoutheastCon 2018*. (04.2018). St. Petersburg, FL : IEEE, 2018. DOI:10.1109/SECON.2018.8478973. P. 1–6.
100. Pan S. J., Yang Q. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*. Vol. 22, Issue 10. P. 1345–1359. DOI:10.1109/TKDE.2009.191.
101. Probst P., Wright M., Boulesteix A.-L. Hyperparameters and Tuning Strategies for Random Forest. 2018. DOI:10.48550/ARXIV.1804.03515.
102. Properties of Naive Bayes. URL: https://nlp.stanford.edu/IR-book/html/htmledition/properties-of-naive-bayes-1.html (accessed 17/10/2025).
103. Richard Maclin, David Opitz. An Empirical Evaluation of Bagging and Boosting. *American Association for Artificial Intelligence*. 1997.
104. Ring M., Wunderlich S., Scheuring D. et al. A Survey of Network-based Intrusion Detection Data Sets. *Computers & Security*. Vol. 86, 09.2019. P. 147–167. DOI:10.1016/j.cose.2019.06.005.
105. Rish I. An Empirical Study of the Naïve Bayes Classifier. *IJCAI 2001 Work Empir Methods Artif Intell*. Vol. 3, 01.01.2001.
106. Rodriguez-Galiano V., Sanchez-Castillo M., Chica-Olmo M. et al. Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines. *Ore Geology Reviews*. Vol. 71, 01.12.2015. P. 804–818. DOI:10.1016/j.oregeorev.2015.01.001.
107. Scheirer W. J., De Rezende Rocha A., Sapkota A. et al. Toward Open Set Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 35, Issue 7. P. 1757–1772. DOI:10.1109/TPAMI.2012.256.
108. Skladannyi P., Kostiuk Y., Rzaeva S. et al. DEVELOPMENT OF MODULAR NEURAL NETWORKS FOR DETECTING DIFFERENT CLASSES OF NETWORK ATTACKS. *Cybersecurity: Education, Science, Technique*. Vol. 3, Issue 27. P. 534–548. DOI:10.28925/2663-4023.2025.27.772.
109. Tavallaee M., Bagheri E., Lu W. et al. A detailed analysis of the KDD CUP 99 data set. *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*. (07.2009). Ottawa, ON, Canada : IEEE, 2009. DOI:10.1109/CISDA.2009.5356528. P. 1–6.
110. Toliupa S., Nakonechnyi V., Uspenskyi O. Signature and statistical analyzers in the cyber attack detection system. *Collection «Information technology and security»*. Vol. 7, Issue 1. P. 69–79. DOI:10.20535/2411-1031.2019.7.1.184326.
111. Toliupa S., Uspenskyi O. Building protection systems on the basis of information multilevel hierarchical model. *Collection «Information technology and security»*. Vol. 4, Issue 2. P. 172–181. DOI:10.20535/2411-1031.2016.4.2.109917.
112. Torralba A., Efros A. A. Unbiased look at dataset bias. *CVPR 20112011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (06.2011). Colorado Springs, CO, USA : IEEE, 2011. DOI:10.1109/CVPR.2011.5995347. P. 1521–1528.

113. Vijaya Saradhi T. A Study on Hyperparameter Tuning in Support Vector Machines and its Impact on Model Accuracy. *Global Journal of Engineering Innovations and Interdisciplinary Research*. Vol. 5, Issue 1. DOI:10.33425/3066-1226.1063.
114. Waagsnes H., Ulltveit-Moe N. Intrusion Detection System Test Framework for SCADA Systems: *Proceedings of the 4th International Conference on Information Systems Security and Privacy4th International Conference on Information Systems Security and Privacy*. (2018). Funchal, Madeira, Portugal : SCITEPRESS - Science and Technology Publications, 2018. DOI:10.5220/0006588202750285. P. 275–285.
115. What Is Random Forest? | IBM. URL: https://www.ibm.com/think/topics/random-forest (accessed 18/10/2025).
116. Why is KNN a lazy learner? URL: https://www.geeksforgeeks.org/machine-learning/why-is-knn-a-lazy-learner/ (accessed 10/10/2025).
117. Zhang H. The Optimality of Naive Bayes. 2004.
118. Zhang X.-G., Yang G.-H. Kullback–Leibler-Divergence-Based Attacks Against Remote State Estimation in Cyber-Physical Systems. *IEEE Transactions on Industrial Electronics*. Vol. 69, Issue 5. P. 5353–5363. DOI:10.1109/TIE.2021.3082073.
119. Ahmad T., Truscan D., Vain J. et al. Early Detection of Network Attacks Using Deep Learning. arXiv, 2022. DOI:10.48550/arXiv.2201.11628.
120. Biau G. Analysis of a Random Forests Model. arXiv, 2012. DOI:10.48550/arXiv.1005.0208.
121. Dewi C., Chen R.-C. Random Forest and Support Vector Machine on Features Selection for Regression Analysis. ICIC International 学会, 2019. DOI:10.24507/ijicic.15.06.2027.
122. Geifman Y., El-Yaniv R. Selective Classification for Deep Neural Networks. arXiv, 2017. DOI:10.48550/arXiv.1705.08500.
123. Islam Y., Chowdhury M. J. U., Das S. C. Advancing Tabular Stroke Modelling Through a Novel Hybrid Architecture and Feature-Selection Synergy. arXiv, 2025. DOI:10.48550/arXiv.2505.15844.
124. Kandadi T. DRAWBACKS OF RANDOM FOREST ALGORITHM TO EXAMINE EXTENSIVE DATASETS. SSRN, 2025. DOI:10.2139/ssrn.5236759.
125. Klusowski J. M. Analyzing CART. arXiv, 2019. DOI:10.48550/ARXIV.1906.10086.
126. Lundberg S., Lee S.-I. A Unified Approach to Interpreting Model Predictions. arXiv, 2017. DOI:10.48550/ARXIV.1705.07874.

ДОДАТКИ

ЗАТВЕРДЖУЮ

Проректор з наукової роботи
та інноваційної діяльності
Національного університету біоресурсів
і природокористування України
доктор сільськогосподарських наук,
професор

Оксана ТОНХА

«17» грудня 2025 р.

ПОГОДЖЕНО

Проректор з науково-педагогічної
роботи та цифрової трансформації
Національного університету біоресурсів
і природокористування України
доктор педагогічних наук, професор

Олена ГЛАЗУНОВА

«17» грудня 2025 р.

А К Т
про впровадження/використання результатів
дисертації на здобуття ступеня доктора філософії
у навчальний процес

Цим актом стверджується, що результати дисертації на тему: «**Інформаційна технологія та інструментальні засоби створення інтелектуальної діагностичної системи захисту інформації**»,

яку представлено на здобуття ступеня доктора філософії
з галузі знань 12 – Інформаційні технології та спеціальності 122 – Комп'ютерні науки,
виконаної Штаньком Вадимом Ігоровичем,

впроваджено у навчальну програму під час викладання дисципліни «Комп'ютерні мережі» (для здобувачів ступеня вищої освіти «Бакалавр») та «Технології побудови захищених комп'ютерних систем» (для здобувачів ступеня вищої освіти «Магістр») у процесі підготовки фахівців зі спеціальності F7 «Комп'ютерна інженерія» у Національному університеті біоресурсів і природокористування України.

У межах дисципліни «Комп'ютерні мережі» використано результати дослідження, що стосуються формалізації мережевих потоків, структуризації трафіку та застосування інтелектуальних методів аналізу даних для моделювання характеристик мережевої взаємодії.

У межах дисципліни «Технології побудови захищених комп'ютерних систем» використано результати дослідження, пов'язані з інтелектуальним аналізом мережевого трафіку, оцінюванням його статистичних характеристик та формалізацією процесів прийняття рішень в умовах неповної або змінної інформації.

Декан факультету інформаційних технологій
д.т.н., професор — Ігор БОЛБОТ

Заступник декана з наукової роботи
д.е.н., професор — Володимир КРАВЧЕНКО

Завідувач кафедри
к.п.н., доцент — Дмитро КАСАТКІН

Додаток Б

Список опублікованих праць за темою дисертації

Стаття у науковому виданні, включеному до міжнародних наукометричних баз даних Scopus та/або Web of Science Core Collection

1. Konyrbaev N., Nikitenko Ye., **Shtanko V.**, Lakhno V., Baishemirov Z., Ibadulla S., Galymzhankyzy A., Myrzabek E. Evaluation and Optimization of the Naive Bayes Algorithm for Intrusion Detection Systems Using the USB-IDS-1 Dataset. Eastern-European Journal of Enterprise Technologies. 2024. Vol. 6. No. 2 (132). P. 74–82. (*Konyrbaev N. сформулював концепцію дослідження та визначив загальну постановку задачі оцінювання ефективності алгоритму Naive Bayes в IDS. Nikitenko Y. визначив наукову гіпотезу щодо залежності ефективності моделі від кількості записів і кількості класів та забезпечив методологічне керівництво дослідженням. Shtanko V. реалізував модель Gaussian Naive Bayes у середовищі Python, виконав підготовку та оброблення даних USB-IDS-1, сформував експериментальний протокол із двома групами розрахунків (залежність від обсягу даних та від кількості класів), здійснив обчислення accuracy та precision, провів регресійний аналіз отриманих результатів і сформулював висновки щодо обмежень алгоритму у багатокласовому режимі. Lakhno V. здійснив інтерпретацію результатів дослідження в контексті практичного застосування систем виявлення вторгнень. Myrzabek E визначив структуру експериментального дизайну та організацію ітераційних обчислень. Ibadulla S. забезпечив формалізацію структури вхідних даних та підготовку наборів для експериментального аналізу. Galymzhankyzy A. виконала статистичну обробку експериментальних результатів та підготовку графічної інтерпретації залежностей. Baishemirov Z. здійснив узагальнення результатів дослідження та підготовку матеріалів до публікації.*).

Стаття у науковому виданні, включеному до категорії «Б» Переліку наукових фахових видань України

2. **Штанько В.**, Нікітенко Є. Проєктування та реалізація віртуального середовища для аналізу мережевого трафіку. Наука і техніка сьогодні. 2025. № 7 (48). С. 2028–2045. (*Штанько В. І. спроєктував архітектуру ізольованого віртуального середовища на базі GNS3 та VirtualBox, реалізував мережеву топологію з використанням маршрутизатора Cisco 2600, налаштував IP-адресацію, маршрутизацію та сегментацію підмереж, розгорнув вузол-«зловмисник» на Kali Linux та вузли-«жертви» на Debian Linux, забезпечив повну ізоляцію віртуальної мережі від зовнішнього середовища, виконав верифікацію топології за допомогою ping, traceroute та аналізу ICMP-пакетів у Wireshark, а також здійснив експериментальну перевірку коректності маршрутизації та мережевої ізоляції. Нікітенко Є. В. сформулював концептуальні засади побудови ізольованого дослідницького середовища для аналізу мережевих атак, визначив методологічні вимоги до контрольованості та відтворюваності експериментальної платформи, а також здійснив науковий супровід і редакційне опрацювання матеріалів*).

Тези наукових доповідей

3. **Штанько В. І.**, Нікітенко Є. В. Актуальні методи побудови систем виявлення вторгнень. Теоретичні та прикладні аспекти розробки комп’ютерних систем ‘2023’: V Всеукраїнська науково-практична конференція студентів і аспірантів, м. Київ, 26 квітня 2023 року: тези доповіді. Київ, 2023. С. 180–181. (*Штанько В. І. здійснив огляд наявних методів побудови систем виявлення вторгнень, виконав аналіз класифікації IDS за методами розгортання та способами виявлення атак, узагальнив переваги й обмеження сигнатурних та аномалійних підходів, а також сформулював висновки щодо доцільності використання сучасних методів у системах захисту інформації. Нікітенко Є. В. визначив концептуальні засади дослідження та здійснив наукове керівництво підготовкою тез.*).

4. Штанько В. І. Порівняльний аналіз навчальних наборів даних для машинного навчання систем виявлення вторгнень. Інформаційні технології: економіка,

техніка, освіта ‘2023’: XIV Міжнародна науково-практична конференція молодих вчених, м. Київ, 26–27 жовтня 2023 року: тези доповіді. Київ, 2023. С. 244–245.
5. Штанько В. І. Використання аналізаторів трафіку для побудови систем виявлення вторгнень. Глобальні та регіональні проблеми інформатизації в суспільстві і природокористуванні ‘2024’: XII Міжнародна науково-практична конференція, м. Київ, 21–22 листопада 2024 року: тези доповіді. Київ, 2024. С. 97–99.
6. Штанько В. І. Швидкодія моделі Naive Bayes з відбором ознак для набору USB-IDS-1. Теоретичні та прикладні аспекти розробки комп’ютерних систем ‘2025’: VII Всеукраїнська науково-практична конференція студентів і аспірантів, м. Київ, 24 квітня 2025 року: тези доповіді. Київ, 2025. С. 375–376.
7. Vadym Shtanko. Usage of GNS3 for cybersecurity research. Achievements of Science and Education in the Modern World. Achievements of Science and Education in the Modern World: 2nd International Scientific Conference, Birmingham, United Kingdom, 14 June 2025: Conference Paper. Birmingham, United Kingdom, 2025. P. 128–130.

Реалізація базового класу для навчання моделей Trainer

```
import gc
import json
import pickle
from copy import copy
from datetime import datetime
from pathlib import Path
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import numpy as np
from sklearn.utils import shuffle
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix,
)
from config import StaticConfig
from custom_data_classes.model_artifacts import TrainingRun
from trainers.common.split_data import SplitData, SplitDataAndLabels
from utils import read_csv_from_folder_to_dataframe


class Trainer:

    def __init__(self, run: TrainingRun):
        print("Initializing Trainer object...")
        self.dataset_path = run.dataset.root
        self.data = []
        self.init_data()

    @staticmethod

    def process_data(x):
        if isinstance(x, int) or isinstance(x, float):
            return x
        elif '-' in x or '*' in x:
            return np.nan
        elif '/' in x:
            return datetime.strptime(x, '%m/%d/%Y %H:%M:%S %p').microsecond
        else:
            return x

    def convert_data(self, data):
        print("Converting data types for all columns...")
        for column in data:
            if column != 'Label':
                data[column] = data[column].apply(self.process_data)
        return data

    def prepare_data(self, data):
        print("Preparing data: converting, cleaning, and shuffling...")
        data = self.convert_data(data)
        data = data.dropna(axis=1, how='all')
        data = data.dropna(axis=0, how='any')
        labels = data["Label"]
        data.drop("Label", axis=1, inplace=True)
        data = data.loc[:, data.var() > .01]
        data["Label"] = labels
        data = shuffle(data)
        return data
```

```
    def clean_excessive_columns(self, data):
        print("Cleaning excessive columns...")
        columns_to_drop = [
            'Flow ID',
            "Src IP",
            "Src Port",
            "Dst IP",
            "Dst Port",
            'predicted',
            'probability_safe',
            'probability_threat',
            'max_confidence',
            'attack',
            'Unnamed: 0',
            'src_mac',
            'dst_mac',
            'Timestamp'
        ]
        for c_id in columns_to_drop:
            if c_id in data.columns:
                data = data.drop([c_id], axis=1)
        return data

    def get_feature_importance(self, model, data):
        raise NotImplemented

    def describe_model(self, model, data):
        feature_importance_df = self.get_feature_importance(model, data)
        feature_importance_df =
feature_importance_df.sort_values(ascending=False)
        feature_importance_df = feature_importance_df.reset_index()
        feature_importance_df.columns = ['Feature', 'Importance']
        feature_importance_df.to_csv(self.classifier.feature_importance,
index=False)
        with open(self.classifier.classifier, "rb") as f:
            model = pickle.load(f)
        with open(self.classifier.description, "wt+") as f1:
            f1.write(json.dumps(model.get_params(), indent=4))
            f1.close()

    def process_files(self, folder_path: Path):
        data = read_csv_from_folder_to_dataframe(folder_path)
        data = self.clean_excessive_columns(data)
        self.data = self.prepare_data(data)

    def init_data(self):
        print("Initializing data from datasets directory...")
        folder_path = self.dataset_path
        self.process_files(folder_path)

    def split_train_calib_test(self, x, y) -> SplitData:
        x_train, x_temp, y_train, y_temp = train_test_split(
            x, y,
            test_size=(1.0 - StaticConfig.split_train),
            random_state=StaticConfig.seed,
            stratify=y
        )
        x_calib, x_test, y_calib, y_test = train_test_split(
            x_temp, y_temp,
            test_size=StaticConfig.split_test / (StaticConfig.split_calib +
StaticConfig.split_test),
            random_state=StaticConfig.seed,
            stratify=y_temp
        )
        del x_temp, y_temp
```

```
        gc.collect()
        return SplitData(x_train, x_calib, x_test, y_train, y_calib, y_test)

    def prepare_data_for_training(self) -> SplitDataAndLabels:
        data_to_check = copy(self.data)
        data = shuffle(data_to_check, random_state=StaticConfig.seed)
        labels = data["Label"].values
        label_encoder = LabelEncoder()
        y = label_encoder.fit_transform(labels)
        with open(self.classifier.labels, "wb") as f:
            pickle.dump(list(label_encoder.classes_), f)
        x = data.drop("Label", axis="columns")
        return SplitDataAndLabels(self.split_train_calib_test(x, y),
label_encoder)

    def calculate_scores(self, y_test, y_pred):
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred, average='macro',
                                    zero_division=np.nan)
        recall = recall_score(y_test, y_pred, average='macro')
        f1 = f1_score(y_test, y_pred, average='macro')
        matrix = confusion_matrix(y_test, y_pred)
        matrix_normalized = confusion_matrix(y_test, y_pred, normalize='true')
        pkl_file = open(self.classifier.labels, 'rb')
        classes = pickle.load(pkl_file)
        pkl_file.close()
        metrics_dict = {
            'accuracy': accuracy,
            'precision': precision,
            'recall': recall,
            'f1': f1,
            'confusion_matrix': matrix.tolist(),
            'confusion_matrix_normalized': matrix_normalized.tolist(),
            'classes': classes,
        }
        path_for_counted_metrics = self.classifier.metrics
        with open(path_for_counted_metrics, "wt+") as f1:
            f1.write(json.dumps(metrics_dict, indent=4))
            f1.close()

    def test_model(self, model, x_test, y_test):
        y_pred = model.predict(x_test)
        self.calculate_scores(y_test, y_pred)
```

Реалізація класу для навчання моделі Наївного Баєса

TrainerNaiveBayes

```
import json
from datetime import datetime
from pathlib import Path
import numpy as np
import gc
import os
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from config import StaticConfig
from custom_data_classes.binary_threshold import BinaryThresholdPayload
from custom_data_classes.model_artifacts import TrainingRun
from trainers.common.split_data import SplitDataAndLabels
from trainers.common.trainer import Trainer
import pickle
from sklearn.naive_bayes import GaussianNB
from binary_threshold_calculator_cost_based import
compute_binary_threshold_tau_b_cost
from utils import parse_training_run_from_cli


class TrainerBayes(Trainer):

    def __init__(self, run: TrainingRun):
        print("Initializing TrainerBayes object with memory optimization...")
        self.data = []
        self.dataset_id = run.dataset.dataset_id
        self.run = run
        self.chunk_size = 10000
        self.classifier = run.model.binary
        self.dataset_path = run.dataset.root
        self.classifier.root.mkdir(parents=True, exist_ok=True)

    @staticmethod
    def simplify_labels(x):
        if x == "BENIGN" or x == 'normative':
            return "normative"
        else:
            return "threat"

    def convert_data_chunk(self, chunk):
        print("Converting data types and simplifying labels for chunk...")
        for column in chunk.columns:
            if column != 'Label':
                chunk[column] = chunk[column].apply(self.process_data)
            else:
                chunk[column] = chunk[column].apply(self.simplify_labels)
        return chunk

    def get_feature_importance(self, model, data):
        mu = model.theta_
        sigma = model.var_
        importance = np.abs(mu[1] - mu[0]) / (
                np.sqrt(sigma[1]) + np.sqrt(sigma[0]) + 1e-9)
        return pd.Series(
            importance,
            index=data.columns,
        )
```

```
    def convert_data(self, data):
        return self.convert_data_chunk(data)

    def process_file(self, file_path):
        print(f"Processing file: {file_path}")
        chunks = []
        for chunk in pd.read_csv(file_path, chunksize=self.chunk_size):
            chunk = self.clean_excessive_columns(chunk)
            chunk = self.convert_data_chunk(chunk)
            chunk = chunk.dropna(axis=1, how='all')
            chunk = chunk.dropna(axis=0, how='any')
            if not chunk.empty:
                chunks.append(chunk)
            gc.collect()
        if not chunks:
            return None
        return pd.concat(chunks, ignore_index=True)

    def init_data(self):
        print("Initializing data from datasets directory with memory
optimization...")
        folder_path = self.dataset_path
        print(f"Loading data from: {folder_path}")
        files = [f for f in os.listdir(folder_path) if f.endswith('.csv')]
        all_data = []
        for file in files:
            file_path = os.path.join(folder_path, file)
            processed_data = self.process_file(file_path)
            if processed_data is not None:
                all_data.append(processed_data)
            gc.collect()
        if not all_data:
            raise ValueError("No valid data found in the dataset files")
        self.data = pd.concat(all_data, ignore_index=True)
        del all_data
        gc.collect()

    def prepare_data_for_training(self) -> SplitDataAndLabels:
        print("Preparing data for training with memory optimization...")
        if not isinstance(self.data, pd.DataFrame) or self.data.empty:
            self.init_data()
        labels = self.data["Label"].values
        labels = pd.Series(labels).apply(self.simplify_labels).values
        data = self.data.drop("Label", axis=1)
        data = data.loc[:, data.var() > .01]
        label_encoder = LabelEncoder()
        encoded_labels = label_encoder.fit_transform(labels)
        if self.classifier.labels is not None:
            with open(self.classifier.labels, 'wb') as output:
                pickle.dump(list(label_encoder.classes_), output)
        print('Data length', len(data))
        return SplitDataAndLabels(self.split_train_calib_test(data,
encoded_labels), label_encoder)

    def test_model(self, model, x_test, y_test):
        print("Testing model with memory optimization...")
        batch_size = 1000
        all_predictions = []
        for i in range(0, len(x_test), batch_size):
            batch = x_test.iloc[i:i + batch_size]
            preds = model.predict(batch)
            all_predictions.extend(preds)
            del batch, preds
            gc.collect()
        y_pred = np.array(all_predictions)
        self.calculate_scores(y_test, y_pred)
```

```
        del y_pred, all_predictions
        gc.collect()

    def describe_model(self, model: GaussianNB, data):
        mu = model.theta_
        sigma = model.var_
        importance = np.abs(mu[1] - mu[0]) / (
                np.sqrt(sigma[1]) + np.sqrt(sigma[0]) + 1e-9)
        feature_importance_nb = pd.Series(
            importance,
            index=data.columns,
        )
        sorted_features = feature_importance_nb.sort_values(ascending=False)
        df = sorted_features.reset_index()
        df.columns = ['Feature', 'Importance']
        df.to_csv(self.classifier.feature_importance, index=True)
        with open(self.classifier.classifier, "rb") as f:
            model = pickle.load(f)
        with open(self.classifier.description, "wt+") as f1:
            f1.write(json.dumps(model.get_params(), indent=4))
            f1.close()

    def train(self, split_data):
        model = GaussianNB(var_smoothing=0.00000001)
        print(f"Model: {model}")
        if len(split_data.x_train) > 50000:
            print("Training in batches to optimize memory usage...")
            model.partial_fit(split_data.x_train.iloc[:10000],
split_data.y_train[:10000],
                              np.unique(split_data.y_train))
            batch_size = 10000
            for i in range(10000, len(split_data.x_train), batch_size):
                end = min(i + batch_size, len(split_data.x_train))
                model.partial_fit(split_data.x_train.iloc[i:end],
split_data.y_train[i:end])
                gc.collect()
        else:
            model.fit(split_data.x_train, split_data.y_train)
        print(f"Saving model to {self.classifier.classifier}")
        with open(self.classifier.classifier, 'wb') as f:
            pickle.dump(model, f)
        return model

    def calibrate_threshold(self, model, split_data, label_encoder):
        normative_code = int(label_encoder.transform(["normative"])[0])
        mask_norm = (split_data.y_calib == normative_code)
        x_calib_norm = split_data.x_calib.loc[mask_norm]
        c_fp = StaticConfig.BINARY_C_FP
        total_payload = []
        for c_fn in StaticConfig.BINARY_C_FN_SCALE:
            tau_b = compute_binary_threshold_tau_b_cost(c_fp=c_fp, c_fn=c_fn)
            model_classes = list(model.classes_)
            threat_code = int(label_encoder.transform(["threat"])[0])
            norm_code = int(label_encoder.transform(["normative"])[0])
            threat_idx = model_classes.index(threat_code)
            norm_idx = model_classes.index(norm_code)
            logp = model.predict_log_proba(x_calib_norm)
            score = (logp[:, threat_idx] - logp[:, norm_idx]).astype(float)
            pass_rate = float(np.mean(score >= tau_b))
            payload = {
                'dataset_id': self.dataset_id,
                "c_fp": float(c_fp),
                "c_fn": float(c_fn),
                "tau_prob": float(c_fp / (c_fp + c_fn)),
                "tau_b": float(tau_b),
                "fpr_on_calib_norm": pass_rate,
```

```
                "calib_norm_size": int(len(x_calib_norm)),
                "classes": list(label_encoder.classes_),
                "split": {"train": StaticConfig.split_train, "calib":
StaticConfig.split_calib,
                          "test": StaticConfig.split_test},
                "seed": int(StaticConfig.seed),
                "timestamp": datetime.now().isoformat(timespec="seconds"),
                "model_path": str(self.classifier.classifier),
                "labels_path": str(self.classifier.labels),
                "score_type": "log_odds",
                "score_definition": "log P(threat|x) - log P(normative|x)"
            }
            total_payload.append(payload)
        path = Path(self.classifier.thresholds / f'thresholds.json')
        path.parent.mkdir(parents=True, exist_ok=True)
        path.write_text(json.dumps(total_payload, ensure_ascii=False, indent=2),
encoding="utf-8")

    def train_and_store(self):
        print("Training Gaussian Naive Bayes model with memory optimization...")
        time_0 = datetime.now()
        res = self.prepare_data_for_training()
        split_data = res.split_data
        time_1 = datetime.now()
        print("Data reading time", time_1 - time_0)
        print(f"Training data shape: {split_data.x_train.shape}")
        print(f"Calib data shape: {split_data.x_calib.shape}")
        print(f"Test data shape: {split_data.x_test.shape}")
        if self.run.only_threshold:
            with open(self.run.model.binary.classifier, 'rb') as f:
                model_data = pickle.load(f)
                if isinstance(model_data, dict):
                    model = model_data['model']
                else:
                    model = model_data
        else:
            model = self.train(split_data)
        print('Training time', datetime.now() - time_1)
        self.calibrate_threshold(model, split_data, res.label_encoder)
        self.test_model(model, split_data.x_test, split_data.y_test)
        self.describe_model(model, split_data.x_test)
        del split_data, model
        gc.collect()


def main():
    run = parse_training_run_from_cli()
    trainer = TrainerBayes(run)
    trainer.train_and_store()
    del trainer
    gc.collect()
    print("Training completed successfully with memory optimization")
if __name__ == "__main__":
    main()
```

Реалізація класу для навчання моделі випадкових лісів TrainerRandomForest

```
import hashlib
import inspect
import os
import gc
import traceback
from datetime import datetime
import numpy as np
import pandas as pd
from sklearn.utils import shuffle
from sklearn.ensemble import RandomForestClassifier
from captor.utils import check_scores, simplify_labels
from config import StaticConfig
from custom_data_classes.model_artifacts import TrainingRun
from trainers.common.trainer import Trainer
import pickle
from utils import parse_training_run_from_cli


class TrainerRandomForest(Trainer):

    def __init__(self, run: TrainingRun):
        self.run = run
        self.result_dir = run.results_dir
        self.classifier = run.model.multiclass
        self.dataset_path = run.dataset.root
        self.dataset_id = run.dataset.dataset_id
        self.classifier.root.mkdir(parents=True, exist_ok=True)
        super().__init__(run)

    def process_files(self, folder_path):
        print(f"Processing CSV files from {folder_path}, excluding files with
'REGULAR' in the name...")
        combined_data = None
        chunk_size = 50000
        files = os.listdir(folder_path)
        files = [file for file in files if file.endswith('.csv')]
        for csv_file in files:
            if csv_file.endswith(".csv"):
                file_path = os.path.join(folder_path, csv_file)
                print(f"Processing {csv_file}...")
                for chunk in pd.read_csv(file_path, chunksize=chunk_size):
                    if 'Label' in chunk.columns:
                        chunk = chunk[~chunk['Label'].isin(['BENIGN',
'normative'])]
                    if chunk.empty:
                        continue
                    chunk = self.clean_excessive_columns(chunk)
                    chunk = shuffle(chunk)
                    chunk = chunk[:min(len(chunk), 50000)]
                    if combined_data is None:
                        combined_data = chunk
                    else:
                        combined_data = pd.concat([combined_data, chunk],
ignore_index=True)
                    gc.collect()
        if combined_data is None:
            print("No data was processed. Check your input files.")
            return
```

```
        combined_data = shuffle(combined_data)
        combined_data = combined_data[:1000000]
        try:
            non_numeric = combined_data.select_dtypes(exclude=["number"])
            print(non_numeric.columns.tolist())
            self.data = self.prepare_data(combined_data)
        except:
            traceback.print_exc()
        gc.collect()

    def convert_data(self, data):
        print("Converting data types for all columns...")
        columns = data.columns.tolist()
        label_idx = columns.index('Label') if 'Label' in columns else -1
        chunk_size = 10
        for i in range(0, len(columns), chunk_size):
            chunk_columns = columns[i:i+chunk_size]
            for column in chunk_columns:
                if column != 'Label':
                    data[column] = data[column].apply(self.process_data)
                else:
                    data[column] = data[column].apply(simplify_labels)
            if 'Label' in chunk_columns:
                data = data[~data['Label'].isin(['BENIGN', 'normative'])]
            gc.collect()
        data = self._optimize_dataframe_memory(data)
        return data

    def get_feature_importance(self, model, data):
        return pd.Series(model.feature_importances_, index=data.columns)

    def train_model(self, split_data):
        model = RandomForestClassifier(
            n_estimators=200,
            random_state=StaticConfig.seed,
            n_jobs=2,
        )
        model.fit(split_data.x_train, split_data.y_train)
        with open(self.classifier.classifier, "wb") as f:
            pickle.dump(model, f)
        return model

    def calibrate_model(self, model, split_data_and_labels):
        split_data = split_data_and_labels.split_data
        X_calib = split_data.x_calib
        expected = model.feature_names_in_
        X_calib = X_calib.reindex(columns=expected, fill_value=0.0)
        proba = model.predict_proba(X_calib)
        pred_idx = model.predict(X_calib)
        label_encoder = split_data_and_labels.label_encoder
        y_true = label_encoder.inverse_transform(split_data.y_calib)
        with open(self.classifier.labels, "rb") as f:
            classes = pickle.load(f)
        max_conf = proba.max(axis=1)
        k_star = np.take(classes, pred_idx)
        df_scores = pd.DataFrame({
            "y_true": y_true,
            "k_star": k_star,
            "p_max": max_conf,
            "is_correct": (k_star == y_true).astype(int),
        })
        out = self.classifier.calibration_data
        os.makedirs(os.path.dirname(out), exist_ok=True)
        df_scores.to_csv(out, index=False)
        now_ts = datetime.now().isoformat(timespec="seconds")
```

```
    def test_model_custom(self, model, split_data_and_labels):
        split_data = split_data_and_labels.split_data
        label_encoder = split_data_and_labels.label_encoder
        X_test = split_data.x_test
        y_true = label_encoder.inverse_transform(split_data.y_test)
        expected = model.feature_names_in_
        X_test = X_test.reindex(columns=expected, fill_value=0.0)
        pred_idx = model.predict(X_test)
        attack_nominal = label_encoder.inverse_transform(pred_idx)
        self.calculate_scores(y_test=y_true, y_pred=attack_nominal)
        results_path = self.result_dir / f'rf_calibration_results_Cmc-
{StaticConfig.MULTI_C_MC}.csv'
        check_scores(
            results_path,
            y_true,
            attack_nominal,
            "multiclass_nominal",
            datetime.now().isoformat(),
            y_true,
            len(y_true),
            label_encoder.classes_,
            {},
            c_fn=None,
        )

    def train_and_store(self):
        print("Training Random Forest model...")
        res = self.prepare_data_for_training()
        split_data = res.split_data
        if self.run.only_threshold:
            with open(self.run.model.multiclass.classifier, 'rb') as f:
                model_data = pickle.load(f)
                if isinstance(model_data, dict):
                    model = model_data['model']
                else:
                    model = model_data
        else:
            model = self.train_model(split_data)
        self.calibrate_model(model, res)
        self.test_model_custom(model, res)

    def _optimize_dataframe_memory(self, df):
        result = df.copy()
        for col in result.select_dtypes(include=['int']).columns:
            result[col] = pd.to_numeric(result[col], downcast='integer')
        for col in result.select_dtypes(include=['float']).columns:
            result[col] = pd.to_numeric(result[col], downcast='float')
        return result


def main():
    run = parse_training_run_from_cli()
    trainer = TrainerRandomForest(run)
    trainer.train_and_store()
if __name__ == "__main__":
    main()
```

Додаток Е

Реалізація основного скрипту для запуску збору і класифікації трафіку

```
import argparse

from captor.flow_capture import FlowCapture


def main():
    parser = argparse.ArgumentParser(description='PyFlowMeter Network Flow
Capture')
    parser.add_argument('-m', '--model_data_id')
    parser.add_argument('-i', '--interface', help='Network interface to capture
from', default='lo')
    parser.add_argument('-w', '--window', type=int, default=60,
                        help='Window size in seconds (default: 60)')
    parser.add_argument('-o', '--output', default='flow_data',
                        help='Output directory (default: flow_data)')
    parser.add_argument('-r', '--root-of-project')

    args = parser.parse_args()

    if args.window < 10:
        print("Warning: Window size less than 10 seconds may not capture enough
flows")

    capture = FlowCapture(
        model_data_id=args.model_data_id,
        interface=args.interface,
        window_size=args.window,
        output_dir=args.output,
        project_root=args.root_of_project
    )

    capture.start_capture()


if __name__ == "__main__":
    main()
```

Реалізація класу для збору трафіку FlowCapture

```
import traceback
import pandas as pd
import time
import signal
import sys
import os
from datetime import datetime
from pyflowmeter.sniffer import create_sniffer
from two_level_classifier.constants import COLUMNS_MAPPING, COLUMNS
from two_level_classifier.data_classifier import DataClassifier


class FlowCapture:

    def __init__(self, model_data_id, interface=None, window_size=60,
output_dir="flow_data", project_root=None):
        self.model_data_id = model_data_id
        self.interface = interface
        self.window_size = window_size
        self.output_dir = output_dir
        self.running = False
        self.window_count = 0
        self.sniffer = None
        self.project_root = project_root
        self.tmp_folder = 'temporary_files'
        os.makedirs(self.tmp_folder, exist_ok=True)
        os.makedirs(self.output_dir, exist_ok=True)
        signal.signal(signal.SIGINT, self._signal_handler)
        signal.signal(signal.SIGTERM, self._signal_handler)

    def _signal_handler(self, signum, frame):
        print(f"\nReceived signal {signum}. Stopping capture...")
        self.stop_capture()
        sys.exit(0)

    def _get_output_filename(self):
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        return os.path.join(self.tmp_folder, f"flows_window_{timestamp}.csv")

    def _get_converted_filename(self, timestamp):
        return os.path.join(self.output_dir, f"flows_window_{timestamp}.csv")

    def _process_window(self, output_file, start_time):
        if os.path.exists(output_file):
            file_size = os.path.getsize(output_file)
            if file_size > 0:
                df = pd.read_csv(output_file)
                if 'flow_id' not in df.columns:
                    df['flow_id'] = df.apply(
                        lambda r: f"{r['src_ip']}:{r['src_port']}-
{r['dst_ip']}:{r['dst_port']}-{r['protocol']}", axis=1
                    )
                col = df.pop('flow_id')
                df.insert(0, 'flow_id', col)
                df['timestamp'] = start_time
                df.rename(columns=COLUMNS_MAPPING, inplace=True)
                available = [c for c in COLUMNS if c in df.columns]
                timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
                df[available].to_csv(self._get_converted_filename(timestamp))
                classifier = DataClassifier(self.model_data_id,
project_root=self.project_root)
```

```
                flows = classifier.classify(df[available], timestamp)
                print(flows)
                print(f"Window {self.window_count} completed - File size:
{file_size} bytes")
            else:
                print(f"Window {self.window_count} completed - No flows
captured")
        else:
            print(f"Window {self.window_count} completed - No output file
created")
        self.window_count += 1

    def _capture_window(self):
        output_file = self._get_output_filename()
        print(f"Starting window {self.window_count} - Output: {output_file}")
        self.sniffer = create_sniffer(
            input_interface=self.interface,
            to_csv=True,
            output_file=output_file,
            verbose=False
        )
        self.sniffer.start()
        start_time = time.time()
        while (time.time() - start_time) < self.window_size and self.running:
            time.sleep(0.1)
        if self.sniffer:
            self.sniffer.stop()
            self.sniffer.join()
        self._process_window(output_file, start_time)

    def start_capture(self):
        print(f"Starting pyflowmeter capture")
        print(f"Interface: {self.interface or 'all interfaces'}")
        print(f"Window size: {self.window_size} seconds")
        print(f"Output directory: {self.output_dir}")
        print("Press Ctrl+C to stop capture\n")
        self.running = True
        try:
            while self.running:
                self._capture_window()
                if self.running:
                    time.sleep(1)
        except Exception as e:
            traceback.print_exc()
            print(f"Error during capture: {e}")
        finally:
            self.stop_capture()

    def stop_capture(self):
        self.running = False
        if self.sniffer:
            try:
                self.sniffer.stop()
                self.sniffer.join()
            except:
                pass
        print("Capture stopped.")
```

Реалізація дворівневого класифікатора DataClassifier

```
import json
import pickle
import logging
import sys
import traceback
from datetime import datetime
from pathlib import Path
import numpy as np
from typing import Dict
import pandas
from pandas import DataFrame
from config import StaticConfig
from custom_data_classes.model_artifacts import DataIds, TwoLevelModelFiles
from captor.utils import check_scores, canonical_labels, simplify_labels,
normalize_dataframe
from two_level_classifier.classifier_utils import (
    load_p0_feature_domain,
    rf_feature_weights,
    compute_s_domain_feature_kl,
    tau_mc_adaptive_from_shift,
)
from utils import add_flow_guid, save_snapshot, save_thin, get_tau_b_by_c_fn


class DataClassifier:

    def __init__(
            self,
            model_data_id: DataIds,
            *,
            models_version: str = StaticConfig.VERSION,
            project_root: Path | None = None,
            results_dir: Path | None = None,
            keep_tmp: bool = False,
    ):
        self.logger = logging.getLogger(__name__)
        self.scaler = None
        self.bundle = TwoLevelModelFiles.from_data_id(
            model_data_id=model_data_id,
            models_version=models_version,
            project_root=project_root,
        )
        self.bundle.ensure(touch_files=False)
        self.binary_model = None
        self.multiclass_model = None
        if results_dir is None:
            root = Path(project_root) if project_root else
Path(__file__).resolve().parent
            results_dir = root / "results" / str(model_data_id)
        self.results_dir = Path(results_dir)
        self.results_dir.mkdir(parents=True, exist_ok=True)
        self.keep_tmp = keep_tmp
        self.load_models()
        self.feature_names = [
            'length', 'protocol', 'ttl', 'flags', 'src_port', 'dst_port',
            'tcp_flags', 'window_size', 'seq_num', 'ack_num', 'duration'
        ]
        self.result_file = f'results.csv'
        binary_thresholds_path = self.bundle.binary.thresholds /
'thresholds.json'
        self.c_fn = 20
```

```
        self.tau_b = get_tau_b_by_c_fn(self.c_fn, binary_thresholds_path)
        self.p0_feature_json_path = Path(self.bundle.multiclass.p0).expanduser()
        self.p0_feature_domain =
load_p0_feature_domain(self.p0_feature_json_path)
        self.rf_weights = rf_feature_weights(self.multiclass_model)

    @staticmethod

    def _load_binary_thresholds_json(path: Path, *, dataset_id: str | None =
None) -> list[dict]:
        obj = json.loads(Path(path).read_text(encoding="utf-8"))
        if not isinstance(obj, list):
            raise ValueError(f"{path}: очікується JSON list, отримано
{type(obj)}")
        rows = []
        for r in obj:
            if not isinstance(r, dict):
                continue
            if dataset_id is not None and str(r.get("dataset_id")) !=
str(dataset_id):
                continue
            if "c_fn" not in r or "tau_b" not in r:
                continue
            rows.append(r)
        if not rows:
            raise ValueError(f"{path}: не знайдено записів з полями c_fn,tau_b
(dataset_id={dataset_id})")
        rows.sort(key=lambda x: float(x["c_fn"]))
        return rows

    def classify_sweep(
            self,
            data: DataFrame,
            timestamp: str,
            *,
            thresholds_json: Path,
            dataset_id: str | None = None,
            c_fn_grid: list[float] | None = None,
    ):
        rows = self._load_binary_thresholds_json(thresholds_json,
dataset_id=dataset_id)
        if c_fn_grid is not None:
            wanted = set(float(x) for x in c_fn_grid)
            rows = [r for r in rows if float(r["c_fn"]) in wanted]
            if not rows:
                raise ValueError("Після фільтрації c_fn_grid не лишилося жодного
порогу в thresholds_json.")
        labels_dataframe = pandas.DataFrame(data, columns=["Label"])
        labels_dataframe["Label"] =
labels_dataframe["Label"].astype("object").fillna("normative")
        y_true =
labels_dataframe["Label"].astype(str).apply(simplify_labels).reset_index(drop=Tr
ue)
        flows_raw = add_flow_guid(data.copy()).reset_index(drop=True).copy()
        flows_raw["_row_id"] = np.arange(len(flows_raw))
        norm_nb = normalize_dataframe(flows_raw)
        X_nb = norm_nb[list(self.binary_model.feature_names_in_)].copy()
        X_nb = X_nb.replace([np.inf, -np.inf], np.nan).fillna(0.0)
        binary_logp = self.binary_model.predict_log_proba(X_nb)
        binary_pred = self.binary_model.predict(X_nb)
        model_classes = list(self.binary_model.classes_)
        classes_bin = self._load_classes(self.bundle.binary.labels)
        label_to_id = {c: i for i, c in enumerate(classes_bin)}
        id_to_label = {i: c for i, c in enumerate(classes_bin)}
        threat_code = label_to_id["threat"]
        norm_code = label_to_id["normative"]
```

```
        if threat_code not in model_classes or norm_code not in model_classes:
            raise ValueError(f"Binary class codes mismatch:
model.classes_={model_classes}")
        threat_idx = model_classes.index(threat_code)
        norm_idx = model_classes.index(norm_code)
        score_log_odds = (binary_logp[:, threat_idx] - binary_logp[:,
norm_idx]).astype(float)
        pred_nominal_bin = np.vectorize(id_to_label.get)(binary_pred)
        y_true_bin = y_true.copy()
        y_true_bin[y_true_bin != "normative"] = "threat"
        n_total = int(flows_raw.shape[0])
        for r in rows:
            c_fn = float(r["c_fn"])
            tau_b = float(r["tau_b"])
            mask_attack_nominal = (pred_nominal_bin == "threat")
            n_attack_nominal_bin = int(np.sum(mask_attack_nominal))
            rate_attack_nominal_bin = (n_attack_nominal_bin / n_total) if
n_total else 0.0
            pred_threshold_bin = np.where(score_log_odds >= tau_b, "threat",
"normative")
            mask_attack = (pred_threshold_bin == "threat")
            n_attack_threshold_bin = int(np.sum(mask_attack))
            rate_attack_threshold_bin = (n_attack_threshold_bin / n_total) if
n_total else 0.0
            times = {}
            check_scores(
                self.results_dir / self.result_file,
                y_true_bin,
                pred_nominal_bin,
                "binary_nominal_sweep",
                timestamp,
                y_true,
                flows_raw.shape[0],
                classes_bin,
                times,
                c_fn=c_fn,
                gated=n_attack_nominal_bin,
                gate_rate=rate_attack_nominal_bin,
            )
            check_scores(
                self.results_dir / self.result_file,
                y_true_bin,
                pred_threshold_bin,
                "binary_threshold_sweep",
                timestamp,
                y_true,
                flows_raw.shape[0],
                classes_bin,
                times,
                threshold=True,
                c_fn=c_fn,
                gated=n_attack_threshold_bin,
                gate_rate=rate_attack_threshold_bin,
                tau_b=tau_b,
            )

    @staticmethod

    def _load_classes(labels_path: Path) -> list[str]:
        with open(labels_path, "rb") as f:
            obj = pickle.load(f)
        return list(obj.classes_) if hasattr(obj, "classes_") else list(obj)

    def load_models(self):
        try:
            with open(self.bundle.binary.classifier, 'rb') as f:
```

```
                model_data = pickle.load(f)
                if isinstance(model_data, dict):
                    self.binary_model = model_data['model']
                    self.scaler = model_data.get('scaler')
                else:
                    self.binary_model = model_data
            with open(self.bundle.multiclass.classifier, 'rb') as f:
                model_data = pickle.load(f)
                if isinstance(model_data, dict):
                    self.multiclass_model = model_data['model']
                else:
                    self.multiclass_model = model_data
            self.logger.info("Models loaded successfully")
        except FileNotFoundError as e:
            self.logger.error(f"Model file not found: {e}")
            raise
        except Exception as e:
            self.logger.error(f"Error loading models: {e}")
            raise

    def preprocess_features(self, features: Dict) -> np.ndarray:
        try:
            feature_vector = []
            for feature_name in self.feature_names:
                if feature_name in features:
                    feature_vector.append(float(features[feature_name]))
                else:
                    feature_vector.append(0.0)
            X = np.array(feature_vector).reshape(1, -1)
            if self.scaler:
                X = self.scaler.transform(X)
            return X
        except Exception as e:
            self.logger.error(f"Error preprocessing features: {e}")
            return np.zeros((1, len(self.feature_names)))

    def process_overall_classification(self, flows, labels, timestamp,
no_suspicious=False, S_domain=None):
        overall = flows.reset_index(drop=True).copy()
        overall["y_true"] = pandas.Series(labels).reset_index(drop=True)
        if "predicted_binary" in overall.columns:
            base_nominal =
overall["predicted_binary"].fillna("normative").astype(str)
        else:
            base_nominal = pandas.Series(["normative"] * len(overall))
        if "predicted_threshold" in overall.columns:
            base_threshold =
overall["predicted_threshold"].fillna("normative").astype(str)
        else:
            base_threshold = pandas.Series(["normative"] * len(overall))
        overall["final_nominal"] = base_nominal.copy()
        overall["final_threshold"] = base_threshold.copy()
        if not no_suspicious:
            has_mc_nom = "attack_nominal" in overall.columns
            has_mc_thr = "attack_threshold" in overall.columns
            if has_mc_nom:
                mask_attack_nominal = (overall["final_nominal"] == "threat")
                if mask_attack_nominal.any():
                    overall.loc[mask_attack_nominal, "final_nominal"] =
overall.loc[mask_attack_nominal, "attack_nominal"]
            if has_mc_thr:
                mask_attack_threshold = (overall["final_threshold"] == "threat")
                if mask_attack_threshold.any():
                    overall.loc[mask_attack_threshold, "final_threshold"] =
overall.loc[
                        mask_attack_threshold, "attack_threshold"
```

```
                    ]
        overall["final_nominal"] =
overall["final_nominal"].fillna("unknown").astype(str)
        overall["final_threshold"] =
overall["final_threshold"].fillna("unknown").astype(str)
        overall["y_true"] =
overall["y_true"].astype("object").fillna("normative").astype(str)
        raw_multi_classes: list[str] =
self._load_classes(self.bundle.multiclass.labels)
        overall_labels = canonical_labels(["normative"] + raw_multi_classes)
        overall_threshold_labels = canonical_labels(["normative"] +
raw_multi_classes + ["unknown"])
        times = {}
        check_scores(
            self.results_dir / self.result_file,
            overall["y_true"],
            overall["final_nominal"],
            "overall_nominal",
            timestamp,
            overall["y_true"],
            overall.shape[0],
            overall_labels,
            times,
            c_fn=self.c_fn,
        )
        check_scores(
            self.results_dir / self.result_file,
            overall["y_true"],
            overall["final_threshold"],
            "overall_threshold_with_unknown",
            timestamp,
            overall["y_true"],
            overall.shape[0],
            overall_threshold_labels,
            times,
            c_fn=self.c_fn,
            s_domain=S_domain
        )
        check_scores(
            self.results_dir / self.result_file,
            overall["y_true"],
            overall["final_threshold"],
            "overall_threshold",
            timestamp,
            overall["y_true"],
            overall.shape[0],
            overall_threshold_labels,
            times,
            threshold=True,
            c_fn=self.c_fn,
            s_domain=S_domain
        )
        if self.keep_tmp:
            path = self.results_dir / f"overall_classification_{timestamp}.csv"
            save_thin(overall, path)
        return overall

    def detect_attacks_with_binary_classifier(self, flows, labels, timestamp,
tau_b, c_fn):
        flows = flows.reset_index(drop=True).copy()
        flows["_row_id"] = np.arange(len(flows))
        norm = normalize_dataframe(flows)
        preprocessed = norm[list(self.binary_model.feature_names_in_)].copy()
        preprocessed = preprocessed.replace([np.inf, -np.inf],
np.nan).fillna(0.0)
        time_before_binary_check = datetime.now()
```

```
        binary_predicted = self.binary_model.predict(preprocessed)
        binary_probability = self.binary_model.predict_proba(preprocessed)
        binary_logp = self.binary_model.predict_log_proba(preprocessed)
        model_classes = list(self.binary_model.classes_)
        classes = self._load_classes(self.bundle.binary.labels)
        id_to_label = {i: c for i, c in enumerate(classes)}
        label_to_id = {c: i for i, c in enumerate(classes)}
        transformed_labels = np.vectorize(id_to_label.get)(binary_predicted)
        threat_code = label_to_id["threat"]
        norm_code = label_to_id["normative"]
        if threat_code not in model_classes or norm_code not in model_classes:
            raise ValueError(
                f"Binary class codes not found in model.classes_. "
                f"model.classes_={model_classes}, threat_code={threat_code},
norm_code={norm_code}"
            )
        threat_idx = model_classes.index(threat_code)
        norm_idx = model_classes.index(norm_code)
        probability_threat = binary_probability[:, threat_idx]
        probability_safe = binary_probability[:, norm_idx]
        flows['predicted_binary'] = transformed_labels
        flows['probability_safe'] = probability_safe
        flows['probability_threat'] = probability_threat
        score_log_odds = (binary_logp[:, threat_idx] - binary_logp[:,
norm_idx]).astype(float)
        flows["score_log_odds"] = score_log_odds
        mask_suspicious = flows["score_log_odds"] >= float(tau_b)
        flows["predicted_threshold"] = np.where(mask_suspicious, "threat",
"normative")
        adjusted_labels = labels.copy().astype("object")
        adjusted_labels[adjusted_labels != 'normative'] = 'threat'
        time_after_binary_check = datetime.now()
        times = {
            'time_binary_classification': time_after_binary_check -
time_before_binary_check,
        }
        writable_result = flows.copy()
        writable_result['labels'] = adjusted_labels
        if self.keep_tmp:
            path = self.results_dir / f"binary_classification_{timestamp}.csv"
            save_thin(writable_result, path)
        flows_all = flows
        labels_all = labels
        flows_suspicious =
flows.loc[mask_suspicious].copy().reset_index(drop=True)
        labels_suspicious = labels.loc[mask_suspicious].reset_index(drop=True)
        has_suspicious = len(flows_suspicious) > 0
        gated_count = int(flows_suspicious.shape[0])
        total_count = int(flows_all.shape[0])
        gate_rate = (gated_count / total_count) if total_count else 0.0
        print("Result of binary classification")
        check_scores(
            self.results_dir / self.result_file,
            adjusted_labels,
            transformed_labels,
            'binary_nominal',
            timestamp,
            labels,
            flows.shape[0],
            classes,
            times,
            c_fn=c_fn,
            gated=gated_count,
            gate_rate=gate_rate,
        )
        check_scores(
```

```
            self.results_dir / self.result_file,
            adjusted_labels,
            flows['predicted_threshold'],
            'binary_threshold',
            timestamp,
            labels,
            flows.shape[0],
            classes,
            times,
            threshold=True,
            c_fn=c_fn,
            gated=gated_count,
            gate_rate=gate_rate,
        )
        return has_suspicious, flows_all, flows_suspicious, labels_all,
labels_suspicious

    def do_multi_classification(self, flows, labels, timestamp,
time_after_data_preprocess, time_before_window_processing):
        time_before_multiclass_check = datetime.now()
        cols_to_drop = [c for c in ['predicted_binary', 'predicted_threshold',
'probability_safe',
                                    'probability_threat']
                        if c in flows.columns]
        stripped_flows = flows.copy().drop(cols_to_drop, axis=1)
        flows = stripped_flows.copy()
        norm = normalize_dataframe(flows)
        preprocessed =
norm[list(self.multiclass_model.feature_names_in_)].copy()
        preprocessed = preprocessed.replace([np.inf, -np.inf],
np.nan).fillna(0.0)
        multiclass_pred = self.multiclass_model.predict(preprocessed)
        multiclass_proba = self.multiclass_model.predict_proba(preprocessed)
        classes = self._load_classes(self.bundle.multiclass.labels)
        model_classes = list(self.multiclass_model.classes_)
        label_map = dict(zip(model_classes,
self._load_classes(self.bundle.multiclass.labels)))
        flows["attack_nominal"] = np.vectorize(label_map.get)(multiclass_pred)
        proba_columns = [f'mc_proba_{c}' for c in classes]
        proba_multiclass_frame = DataFrame(multiclass_proba,
columns=proba_columns)
        p_max = multiclass_proba.max(axis=1)
        S_domain, df_dom_detail = compute_s_domain_feature_kl(
            current_df=flows,
            p0_domain=self.p0_feature_domain,
            rf_weights=self.rf_weights,
            eps=1e-12,
            divergence="kl",
        )
        tau_mc = tau_mc_adaptive_from_shift(
            s_domain=S_domain,
        )
        flows.loc[:, "p_max"] = p_max
        flows.loc[:, "tau_mc"] = float(tau_mc)
        flows.loc[:, "S_domain"] = float(S_domain)
        flows.loc[:, "attack_threshold"] = np.where(
            p_max >= float(tau_mc),
            flows["attack_nominal"],
            "unknown",
        )
        classified_flows = pandas.concat(
            [flows.reset_index(drop=True),
proba_multiclass_frame.reset_index(drop=True)],
            axis=1
        )
        unknown_rate =
```

```
float(np.mean(classified_flows["attack_threshold"].to_numpy() == "unknown")) if
len(classified_flows) else 0.0
        if self.keep_tmp:
            path = self.results_dir /
f"multiclass_classification_{timestamp}.csv"
            save_thin(classified_flows, path)
        print("Some suspicious or dangerous flows were detected")
        time_after_multiclass_check = datetime.now()
        times = {
            'time_window_preprocess': time_after_data_preprocess -
time_before_window_processing,
            'time_multiclass_classification': time_after_multiclass_check -
time_before_multiclass_check,
            'time_inference': time_after_multiclass_check -
time_before_window_processing,
        }
        mc_labels_nominal = canonical_labels(["normative"] + classes)
        mc_labels_threshold = canonical_labels(["normative"] + classes +
["unknown"])
        check_scores(
            self.results_dir / self.result_file,
            labels.astype("object").fillna("normative").astype(str),
            classified_flows['attack_nominal'],
            'multiclass_nominal',
            timestamp,
            labels,
            classified_flows.shape[0],
            mc_labels_nominal,
            times,
        )
        check_scores(
            self.results_dir / self.result_file,
            labels.astype("object").fillna("normative").astype(str),
            classified_flows['attack_threshold'],
            'multiclass_threshold_with_unknown',
            timestamp,
            labels,
            classified_flows.shape[0],
            mc_labels_threshold,
            times,
            s_domain=S_domain,
            unknown_rate=unknown_rate,
            tau_mc=tau_mc,
        )
        check_scores(
            self.results_dir / self.result_file,
            labels.astype("object").fillna("normative").astype(str),
            classified_flows['attack_threshold'],
            'multiclass_threshold',
            timestamp,
            labels,
            classified_flows.shape[0],
            mc_labels_threshold,
            times,
            threshold=True,
            s_domain=S_domain,
            unknown_rate=unknown_rate,
            tau_mc=tau_mc,
        )
        return classified_flows

    def classify(self, data: DataFrame, timestamp: str):
        try:
            time_before_window_processing = datetime.now()
            labels_dataframe = pandas.DataFrame(data, columns=["Label"])
            labels_dataframe["Label"] =
```

```
labels_dataframe["Label"].astype("object").fillna("normative")
            retrieved_labels =
labels_dataframe["Label"].astype(str).apply(simplify_labels)
            flows_raw = add_flow_guid(data.copy())
            if self.keep_tmp:
                path = self.results_dir / f"snapshot_{timestamp}.csv"
                save_snapshot(flows_raw, path)
            time_after_data_preprocess = datetime.now()
            if self.binary_model:
                has_suspicious, flows_all, flows_suspicious, labels_all,
labels_suspicious = \
                    self.detect_attacks_with_binary_classifier(
                        flows_raw,
                        retrieved_labels,
                        timestamp,
                        tau_b=self.tau_b,
                        c_fn=self.c_fn,
                    )
                if has_suspicious and self.multiclass_model:
                    rf_scored = self.do_multi_classification(
                        flows_suspicious,
                        labels_suspicious,
                        timestamp,
                        time_after_data_preprocess,
                        time_before_window_processing
                    )
                    rf_cols = ["_row_id", "attack_nominal", "attack_threshold",
"p_max", "tau_mc", "S_domain"]
                    existing = [c for c in rf_cols if c in rf_scored.columns]
                    flows_all = flows_all.merge(
                        rf_scored[existing],
                        on="_row_id",
                        how="left",
                    )
                    overall = self.process_overall_classification(flows_all,
labels_all, timestamp)
                    return overall
                S_domain, df_dom_detail = compute_s_domain_feature_kl(
                    current_df=flows_all,
                    p0_domain=self.p0_feature_domain,
                    rf_weights=self.rf_weights,
                    eps=1e-12,
                    divergence="kl",
                )
                if "Label" not in flows_all.columns:
                    flows_all["Label"] = "normative"
                flows_all["Label"] = flows_all["Label"].astype("object")
                labels_all = labels_all.astype("object")
                flows_all.loc[:, "Label"] = labels_all
                if self.keep_tmp:
                    path = self.results_dir / f"no_attacks_{timestamp}.csv"
                    save_thin(flows_all, path)
                overall = self.process_overall_classification(flows_all,
labels_all, timestamp, True, S_domain)
                return overall
            raise RuntimeError("Binary model is not loaded")
        except Exception as e:
            ex_type, e, tb = sys.exc_info()
            print(e)
            traceback.print_tb(tb)
            raise
```

Реалізація допоміжних функцій для обчислення оцінок доменного зсуву та адаптивних порогів

```
from __future__ import annotations
import numpy as np
from config import StaticConfig
import json
from pathlib import Path
from typing import Dict, Tuple, Optional
import pandas as pd

def rf_feature_weights(rf_model) -> Dict[str, float]:
    names = list(getattr(rf_model, "feature_names_in_", []))
    imps = np.asarray(getattr(rf_model, "feature_importances_", []),
dtype=float)
    if len(names) == 0 or imps.size == 0:
        return {}
    imps = np.clip(imps, 0.0, None)
    s = float(imps.sum())
    if s <= 0.0:
        w = 1.0 / max(1, len(names))
        return {n: float(w) for n in names}
    imps = imps / s
    return {n: float(w) for n, w in zip(names, imps)}

def _normalize_prob_from_counts(counts: np.ndarray) -> np.ndarray:
    counts = np.asarray(counts, dtype=np.float64)
    s = float(np.sum(counts))
    if s <= 0.0:
        return np.zeros_like(counts, dtype=np.float64)
    return counts / s

def kl_eps_den(pA: np.ndarray, pB: np.ndarray, eps: float) -> float:
    pA = np.asarray(pA, dtype=np.float64)
    pB = np.asarray(pB, dtype=np.float64)
    mask = pA > 0
    if not np.any(mask):
        return 0.0
    return float(np.sum(pA[mask] * np.log(pA[mask] / (pB[mask] + float(eps)))))

def sym_kl_eps_den(pA: np.ndarray, pB: np.ndarray, eps: float) -> float:
    return 0.5 * (kl_eps_den(pA, pB, eps) + kl_eps_den(pB, pA, eps))

def load_p0_feature_domain(path: Path) -> dict:
    path = Path(path).expanduser().resolve()
    with open(path, "r", encoding="utf-8") as f:
        obj = json.load(f)
    if not isinstance(obj, dict) or "features" not in obj:
        raise ValueError(f"{path}: expected a dictionary with key 'features'")
    if not isinstance(obj["features"], list):
        raise ValueError(f"{path}: 'features' must be a list")
    return obj

def _extract_p0_maps(p0_domain: dict) -> Tuple[Dict[str, np.ndarray], Dict[str,
np.ndarray]]:
    edges_map: Dict[str, np.ndarray] = {}
    p0_prob_map: Dict[str, np.ndarray] = {}
    for item in p0_domain.get("features", []):
        feat = str(item.get("feature", ""))
        if not feat:
            continue
```

```
        edges = np.asarray(item.get("edges", []), dtype=np.float64)
        prob = np.asarray(item.get("prob", []), dtype=np.float64)
        if edges.size < 3 or prob.size < 2:
            continue
        if edges.size != prob.size + 1:
            raise ValueError(
                f"Bin mismatch for feature='{feat}': len(edges)={edges.size},
len(prob)={prob.size}"
            )
        edges_map[feat] = edges
        p0_prob_map[feat] = prob
    if not edges_map:
        raise ValueError("P0: no valid features with edges/prob found.")
    return edges_map, p0_prob_map

def compute_s_domain_feature_kl(
    *,
    current_df: pd.DataFrame,
    p0_domain: dict,
    rf_weights: Optional[Dict[str, float]] = None,
    eps: float = 1e-12,
    divergence: str = "kl",
) -> Tuple[float, pd.DataFrame]:
    edges_map, p0_prob_map = _extract_p0_maps(p0_domain)
    feats = sorted(p0_prob_map.keys())
    if rf_weights:
        w_raw = {f: float(rf_weights.get(f, 0.0)) for f in feats}
        s = float(sum(w_raw.values()))
        if s <= 0.0:
            w = 1.0 / max(1, len(feats))
            weights = {f: w for f in feats}
        else:
            weights = {f: (w_raw[f] / s) for f in feats}
    else:
        w = 1.0 / max(1, len(feats))
        weights = {f: w for f in feats}
    rows = []
    S = 0.0
    for f in feats:
        if f not in current_df.columns:
            continue
        edges = edges_map[f]
        p0 = p0_prob_map[f]
        x = pd.to_numeric(current_df[f],
errors="coerce").to_numpy(dtype=np.float64)
        x = x[np.isfinite(x)]
        if x.size == 0:
            continue
        counts, _ = np.histogram(x, bins=edges)
        pt = _normalize_prob_from_counts(counts)
        if divergence == "kl":
            d = kl_eps_den(p0, pt, eps=eps)
        elif divergence == "symkl":
            d = sym_kl_eps_den(p0, pt, eps=eps)
        else:
            raise ValueError("divergence must be 'kl' or 'symkl'")
        w = float(weights.get(f, 0.0))
        S += w * float(d)
        rows.append({"feature": f, "w": w, "D": float(d), "wD": float(w * d)})
    df_detail = pd.DataFrame(rows).sort_values("wD",
ascending=False).reset_index(drop=True)
    return float(S), df_detail

def tau_mc_adaptive_from_shift(*, s_domain: float) -> float:
    c_reject = StaticConfig.C_REJECT
    c_misclass = StaticConfig.C_MISCLASS
```

```
beta = StaticConfig.beta
clip_min = 0.0
clip_max = 1.0
s = float(max(0.0, s_domain))
denom = float(c_misclass) * (1.0 + float(beta) * s)
if denom <= 0.0:
    return float(clip_max)
tau = 1.0 - (float(c_reject) / denom)
return float(np.clip(tau, clip_min, clip_max))
```

Реалізація допоміжних функцій для обчислення та збереження метрик

```
import csv
import os
from datetime import datetime
from pathlib import Path
from pandas import DataFrame, Series
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix,
)


def simplify_labels(x):
    if x == "BENIGN" or x == 'normative':
        return "normative"
    if '-' in x:
        split = x.split('-')
        if len(split) > 1:
            return split[0]
    return x


def process_data(x):
    if isinstance(x, int) or isinstance(x, float):
        return x
    elif '-' in x or '*' in x:
        return np.nan
    elif '/' in x:
        return datetime.strptime(x, '%m/%d/%Y %H:%M:%S %p').timestamp()
    else:
        return x


def normalize_dataframe(data):
    copied = DataFrame()
    for column in data:
        if column != 'Label' and column != 'Flow ID':
            copied[column] = data[column].copy().apply(process_data)
        if column == 'Flow ID':
            copied[column] = data[column].copy()
    copied = copied.replace([np.inf, -np.inf], np.nan)
    copied = copied.fillna(0.0)
    return copied
import numpy as np


def _safe_div(a: np.ndarray, b: np.ndarray) -> np.ndarray:
    a = np.asarray(a, dtype=float)
    b = np.asarray(b, dtype=float)
    out = np.full_like(a, np.nan, dtype=float)
    mask = b != 0.0
    out[mask] = a[mask] / b[mask]
    return out


def confusion_stats_from_cm(cm: np.ndarray, labels: list[str]) -> dict:
    cm = np.asarray(cm, dtype=np.int64)
```

```
    if cm.ndim != 2 or cm.shape[0] != cm.shape[1]:
        raise ValueError(f"confusion matrix must be square, got
shape={cm.shape}")
    k = cm.shape[0]
    total = cm.sum()
    tp = np.diag(cm)
    fp = cm.sum(axis=0) - tp
    fn = cm.sum(axis=1) - tp
    tn = total - tp - fp - fn
    tpr = _safe_div(tp, tp + fn)
    tnr = _safe_div(tn, tn + fp)
    fpr = _safe_div(fp, fp + tn)
    fnr = _safe_div(fn, fn + tp)
    ppv = _safe_div(tp, tp + fp)
    npv = _safe_div(tn, tn + fn)

    def _macro(x: np.ndarray) -> float:
        return float(np.nanmean(x)) if x.size else float("nan")
    return {
        "labels": list(labels),
        "total": int(total),
        "tp": tp.astype(int).tolist(),
        "tn": tn.astype(int).tolist(),
        "fp": fp.astype(int).tolist(),
        "fn": fn.astype(int).tolist(),
    }


def check_scores(
    results_file,
    y_test,
    y_pred,
    classification,
    timestamp,
    all_labels,
    flows_number,
    possible_labels,
    times=None,
    *,
    threshold: bool = False,
    unknown_label: str = "unknown",
    c_fn = None,
    gated = None,
    gate_rate = None,
    s_domain = None,
    tau_mc = None,
    unknown_rate = None,
    tau_b = None,
):
    if times is None:
        times = {}
    y_test_arr = np.asarray(y_test, dtype=object)
    y_pred_arr = np.asarray(y_pred, dtype=object)
    if threshold:
        n_total = int(len(y_pred_arr))
        accepted_mask = (y_pred_arr != unknown_label)
        n_accepted = int(accepted_mask.sum())
        coverage = float(n_accepted / n_total) if n_total > 0 else 0.0
        y_test_eval = y_test_arr[accepted_mask]
        y_pred_eval = y_pred_arr[accepted_mask]
        possible_labels = [lbl for lbl in possible_labels if lbl !=
unknown_label]
        if len(y_pred_eval) > 0:
            accuracy = accuracy_score(y_test_eval, y_pred_eval)
            precision = precision_score(y_test_eval, y_pred_eval,
average="macro", zero_division=np.nan)
```

```
            recall = recall_score(y_test_eval, y_pred_eval, average="macro")
            f1 = f1_score(y_test_eval, y_pred_eval, average="macro")
            c_matrix = confusion_matrix(y_test_eval, y_pred_eval,
labels=possible_labels)
            stats = confusion_stats_from_cm(c_matrix, possible_labels)
        else:
            accuracy = np.nan
            precision = np.nan
            recall = np.nan
            f1 = np.nan
            c_matrix = np.zeros((len(possible_labels), len(possible_labels)),
dtype=int)
            stats = confusion_stats_from_cm(c_matrix, possible_labels)
    else:
        n_total = int(len(y_pred_arr))
        n_accepted = n_total
        coverage = 1.0 if n_total > 0 else 0.0
        unknown_rate = 0.0
        accuracy = accuracy_score(y_test_arr, y_pred_arr)
        precision = precision_score(y_test_arr, y_pred_arr, average="macro",
zero_division=np.nan)
        recall = recall_score(y_test_arr, y_pred_arr, average="macro")
        f1 = f1_score(y_test_arr, y_pred_arr, average="macro")
        c_matrix = confusion_matrix(y_test_arr, y_pred_arr,
labels=possible_labels)
        stats = confusion_stats_from_cm(c_matrix, possible_labels)
    write_header = not os.path.isfile(results_file) or
os.path.getsize(results_file) == 0
    field_names = [
        "timestamp",
        "classification",
        "number_of_flows",
        "flows_variance",
        "confusion_matrix",
        "accuracy",
        "precision",
        "recall",
        "f1",
        "tp", "tn", "fp", "fn",
        "coverage",
        "unknown_rate",
        "n_total",
        "n_accepted",
        "flows_number",
        "all_labels",
        "time_window_preprocess",
        "time_binary_classification",
        "time_multiclass_classification",
        "time_inference",
        'c_fn',
        'gated',
        'gate_rate',
        's_domain',
        'tau_mc',
        'unknown_rate',
        'tau_b',
    ]
    results_file = Path(results_file)
    results_file.parent.mkdir(parents=True, exist_ok=True)
    with open(results_file, mode="a", newline="") as f:
        writer = csv.DictWriter(f, fieldnames=field_names)
        if write_header:
            writer.writeheader()
        labels_series = Series(all_labels)
        to_write = {
            "timestamp": timestamp,
```

```
            "classification": classification,
            "number_of_flows": n_total,
            "flows_variance": Series(y_test_arr).value_counts().to_dict(),
            "confusion_matrix": c_matrix.tolist(),
            "accuracy": float(accuracy) if accuracy == accuracy else accuracy,
            "precision": float(precision) if precision == precision else
precision,
            "recall": float(recall) if recall == recall else recall,
            "f1": float(f1) if f1 == f1 else f1,
            "coverage": coverage,
            "n_total": n_total,
            "n_accepted": n_accepted,
            "all_labels": list(possible_labels),
            "tp": stats["tp"],
            "tn": stats["tn"],
            "fp": stats["fp"],
            "fn": stats["fn"],
            "c_fn": c_fn,
            "gated": gated,
            "gate_rate": gate_rate,
            's_domain': s_domain,
            'tau_mc': tau_mc,
            'unknown_rate': unknown_rate,
            'tau_b': tau_b,
        }
        if times:
            to_write.update(times)
        writer.writerow(to_write)


def canonical_labels(labels, *, normative_label="normative",
unknown_label='unknown'):
    seen = set()
    out = []
    if normative_label in labels:
        out.append(normative_label)
        seen.add(normative_label)
    for lbl in labels:
        if lbl not in seen and lbl != unknown_label:
            out.append(lbl)
            seen.add(lbl)
    if unknown_label and unknown_label in labels:
        out.append(unknown_label)
    return out
```