Міністерство освіти і науки України
Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра інформаційної та кібернетичної безпеки
імені професора Володимира Бурячка

«Допущено до захисту»
Завідувач кафедри інформаційної та
кібернетичної безпеки імені
професора Володимира Бурячка
кандидат технічних наук, доцент
Складанний П.М.

_____
(підпис)
« ___ » _____ 20__ р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття другого (магістерського)
рівня вищої освіти

Спеціальність 125 Кібербезпека та захист інформації

Тема роботи:
СИСТЕМА КІБЕРБЕЗПЕКИ НА БАЗІ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ МОНІТОРИНГУ ТА ВИЯВЛЕННЯ ВТОРГНЕНЬ У МЕРЕЖЕВОМУ ТРАФІКУ

Виконав
студент групи

Непомнящий К.М.
(прізвище, ім’я, по батькові)

_____
(підпис)

Науковий керівник
доцент
(науковий ступінь, наукове звання)
Аносов А.О.
(прізвище, ініціали)

_____
(підпис)

Київ – 2025

Міністерство освіти і науки України
Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра інформаційної та кібернетичної безпеки
імені професора Володимира Бурячка

Освітньо-кваліфікаційний рівень – магістр
Спеціальність 125 Кібербезпека та захист інформації
Освітня програма 125.00.01 Безпека інформаційних і комунікаційних систем

«Затверджую»
Завідувач кафедри інформаційної та
кібернетичної безпеки імені
професора Володимира Бурячка
кандидат технічних наук, доцент
Складанний П.М.

_____
(підпис)
« ___ » _____ 20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

__Непомящему Костянтину Михайловичу__
(прізвище, ім'я, по батькові)

1. Тема роботи: Система кібербезпеки на базі штучного інтелекту для моніторингу та виявлення вторгнень у мережевому трафіку;
   керівник Аносов А.О., к.в.н, доцент
   затверджені наказом ректора від «__» _____ 2024 року №__.
2. Термін подання студентом роботи «__» _____ 2025 р.
3. Вихідні дані до роботи: міжнародні та національні нормативно-правові документи у сфері кібербезпеки, відкритий набір даних NSL-KDD, розміщений на платформі Kaggle, що використовується для навчання та тестування систем виявлення вторгнень, офіційні статистичні звіти щодо інцидентів інформаційної безпеки у корпоративних і державних мережах, наукові статті та аналітичні дослідження провідних українських і зарубіжних фахівців з машинного навчання та систем виявлення аномалій у мережевому трафіку.
4. Зміст текстової частини роботи (перелік питань, які потрібно розробити):
   4.1. Аналітичний огляд проблеми та вибір напрямків дослідження.

4.2. Теоретичні та експериментальні методи дослідження.

4.3. Аналіз, узагальнення та інтерпретація результатів.

5. Перелік графічного матеріалу:

5.1. Презентація доповіді, виконана в Microsoft PowerPoint.

6. Дата видачі завдання «__» _____ 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів підготовки роботи	Термін виконання	Примітка
1.	Уточнення постановки завдання		
2.	Аналіз літератури		
3.	Обґрунтування вибору рішення		
4.	Збір даних		
5.	Виконання та оформлення розділу 1.		
6.	Виконання та оформлення розділу 2.		
7.	Виконання та оформлення розділу 3.		
8.	Вступ, висновки, реферат		
9.	Апробація роботи на науково-методичному семінарі та науково-технічній конференції		
10.	Оформлення та друк текстової частини роботи		
11.	Оформлення презентацій		
12.	Отримання рецензій		
13.	Попередній захист роботи		
14.	Захист в ЕК		

Студент _____
(прізвище, ім'я, по батькові)

Науковий керівник _____
(прізвище, ім'я, по батькові)

РЕФЕРАТ

Кваліфікаційна робота присвячена розробленню інтелектуальної системи для виявлення кіберінцидентів у мережевому трафіку з використанням алгоритмів машинного навчання.

Робота складається зі вступу, трьох розділів, які містять 34 рисунків і 9 таблиць, висновків та списку використаних джерел із 48 найменувань. Загальний обсяг становить 98 сторінки, з яких 7 сторінок займають ілюстрації та таблиці на окремих аркушах, а також додатки, перелік умовних скорочень і список використаних джерел. У роботі виконано аналіз сучасних методів виявлення вторгнень, побудовано математичну модель класифікації мережевого трафіку, розроблено програмний модуль для автоматичного виявлення атак і проведено експериментальну оцінку точності моделі.

Об'єкт дослідження – процес моніторингу та аналізу мережевого трафіку з метою виявлення і класифікації кіберзагроз у системах інформаційної безпеки.

Предмет дослідження – методи, алгоритми та програмні засоби застосування штучного інтелекту для автоматичного виявлення вторгнень і підвищення ефективності систем моніторингу мережевої активності.

Мета дослідження полягає у розробці інтелектуальної системи кібербезпеки, здатної здійснювати моніторинг мережевого трафіку в реальному часі та автоматично виявляти вторгнення на основі алгоритмів машинного навчання, забезпечуючи підвищення точності класифікації загроз і зменшення кількості хибних спрацьовувань.

Для досягнення поставленої мети необхідно вирішити такі основні задачі дослідження:

– проаналізувати літературні джерела та визначити основні концепції мережевої безпеки;

– обґрунтувати вибір алгоритмів машинного навчання для задачі класифікації мережевого трафіку;

– розробити інтелектуальну систему моніторингу, що включає етапи

підготовки даних, навчання моделі, її оцінювання та інтеграцію у процес реального аналізу мережевих потоків;

– провести експериментальну перевірку ефективності моделі, здійснити аналіз отриманих результатів.

Наукова новизна одержаних результатів. Наукова новизна полягає у розробленні інтелектуальної системи моніторингу мережевого трафіку, яка поєднує методи машинного навчання з механізмами автоматичного виявлення вторгнень. Запропоновано комплексний підхід до інтеграції числових і категорійних ознак трафіку в єдину модель класифікації, що забезпечує підвищену точність і стійкість до змін мережевого середовища.

Галузь застосування. Результати дослідження можуть бути використані у підрозділах кібербезпеки підприємств, державних органів, фінансових структур і організацій критичної інфраструктури, а також у навчальному процесі підготовки фахівців з інформаційної безпеки та аналізу даних.

Ключові слова: ВИЯВЛЕННЯ ВТОРГНЕНЬ, КІБЕРІНЦИДЕНТИ, МЕРЕЖЕВИЙ ТРАФІК, МАШИННЕ НАВЧАННЯ, АНАЛІЗ АНОМАЛІЙ, IDS, AI CYBERMONITOR, БЕЗПЕКА ІНФОРМАЦІЙНИХ СИСТЕМ.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ІКС – інформаційно-комунікаційні системи

КІ – критична інфраструктура

МН – машинне навчання

CTI – Cyber Threat Intelligence

IDS – системи виявлення вторгнень

LUFlow – Lancaster University Flow

SIEM – системи управління подіями безпеки

SVD – Singular Value Decompozition

ВСТУП

У сучасному інформаційному суспільстві розвиток цифрових технологій та безперервне зростання обсягів мережевих комунікацій створюють надзвичайно сприятливе середовище для виникнення кіберзагроз. Зловмисні суб’єкти постійно вдосконалюють свої інструменти – від простих сканерів портів до складних мережевих атак, що застосовують методи соціальної інженерії, ботнети, розподілені DDoS-атаки, експлойти нульового дня тощо. Традиційні системи виявлення вторгнень, засновані на сигнатурах або правилових підходах, дедалі частіше виявляються недостатньо гнучкими: вони не справляються з новими, невідомими варіантами атак і мають значні показники помилок.

У відповідь на ці виклики у науковій спільноті зростає інтерес до впровадження систем кібербезпеки, підсилених методами штучного інтелекту (ШІ). Такі системи спроможні аналізувати великі обсяги мережевого трафіку в реальному часі, виявляти аномалії та передбачати потенційні загрози на основі статистичних закономірностей та моделей поведінки. У роботі [1] автори підкреслюють, що хоча застосування ШІ відкриває можливості для адаптивного виявлення загроз і автоматизації безпекових процесів, воно також стикається з такими проблемами як високий рівень помилкових спрацьовувань, дисбаланс класів, атаки-протиагенти й необхідність безперервного оновлення моделі.

У статті [2] висвітлено проблему прозорості й інтерпретованості моделей машинного навчання у системах безпеки: використання методів пояснень дозволяє підвищити довіру аналітиків до рішень, зменшити ризики непродуманих реакцій і забезпечити контрольованість рішень. Інша робота [3] у свою чергу показує, що у контексті інтелектуальних інфраструктур (IoT, індустріальні системи, «розумні міста») виявлення кіберзагроз на основі ШІ стає не лише бажаним, а й життєво необхідним через високу вразливість таких систем до мережевих атак.

Отже, існує нагальна потреба у розробці систем, які поєднують точність, адаптивність і пояснюваність. Розробка такої системи потребує вирішення численних наукових завдань: оптимального добору ознак для класифікації трафіку,

побудови моделей з балансом між чутливістю та варіативністю, забезпечення стійкості проти протидій зі сторони зловмисників, а також інтеграції елементів пояснення в аналітичний процес.

Оцінка сучасного стану проблеми на основі вітчизняної та зарубіжної літератури. У вітчизняному просторі дослідження в області виявлення мережевих вторгнень із застосуванням методів машинного навчання (МН) та штучного ШІ набувають усе більшої актуальності. Наприклад, у роботі Нікітенка висвітлюється дослідження алгоритмів МН для побудови систем виявлення вторгнень, що аналізують мережевий трафік із метою класифікації підозрілих подій [4]. В іншій публікації [5] того самого автора розглянуто системи виявлення вторгнень на основі нейронних мереж глибокого навчання – підкреслюється, що глибинні моделі можуть ефективно захоплювати складні нелінійні залежності у мережевих потоках, проте потребують ретельного налаштування й ресурсів обчислювальної інфраструктури.

У журналі «C-Security» опубліковано дослідження Є. Чичкарьова «Виявлення мережевих вторгнень з використанням методу машинного навчання з вибором ознак у великих наборах даних», в якому авторами запропоновано підхід із відбором ознак за допомогою статистичних тестів і нечітких правил, а також порівняно продуктивність восьми алгоритмів машинного навчання. За результатами показано, що запропонована система скорочує час виявлення до 60 % при високій точності класифікації атак [6].

Отже, у вітчизняній літературі вже присутні конкретні приклади застосування МН/ШІ-підходів у задачах виявлення мережевих атак, зокрема з відбором ознак, глибинним навчанням і експериментальними порівняннями алгоритмів. Разом із тим, ці роботи, як правило, зосереджені на окремих алгоритмах або демонструють ефективність на обмежених наборах даних, без масштабного дослідження питань масштабованості, стійкості до атак-протиагентів і інтеграції в реальні мережеві середовища.

У зарубіжній літературі тема застосування ШІ у системах виявлення вторгнень розвивається дуже інтенсивно й охоплює як оглядові роботи, так і конкретні

інноваційні підходи. У роботі [7] здійснено систематичний аналіз понад 70 публікацій, що охоплюють методи ML, DL і ансамблеві моделі; автори підкреслюють, що AI-підходи значно покращують точність, але при цьому багато дослідників звертають увагу здебільшого на загальний рівень ефективності, а не на характеристики розрізнення окремих класів атак. У статті [8] розглянуто питання оптимізації моделей для обмежених обчислювальних ресурсів, компроміси між продуктивністю та складністю, а також виклики застосування AI у вбудованих або мережевих середовищах із обмеженими ресурсами.

Публікація [9] – аналізує сучасні AI-методи у кібербезпеці, порівнює їхні можливості та обмеження. Зокрема, автори вказують на актуальність постійного оновлення моделей, адаптацію до нових загроз та потребу враховувати гібридні сценарії атак. Також, у дослідженні [10] висвітлено питання інтеграції методів пояснюваного ШІ (XAI) у системи виявлення вторгнень: автори показують, що застосування підходів пояснюваності (наприклад, LIME, SHAP) дозволяє збільшити прозорість рішень моделей, підвищити довіру фахівців і знизити ризики некоректної інтерпретації результатів.

На основі цього можна зробити такі узагальнення та критичні висновки:

– здобутки. Зарубіжні та вітчизняні дослідження підтверджують, що застосування методів ШІ (ML, DL, ансамблі) у задачах виявлення мережевих атак забезпечує відчутні переваги щодо точності та гнучкості порівняно з класичними сигнатурними чи правиловими підходами;

– обмеження. Більшість досліджень використовують стандартні (історичні) набори даних (NSL-KDD, CICIDS, UNSW-NB15 тощо), які не відображають сучасні атаки; нерідко не оцінюється стійкість моделей до атак-протиагентів; питання масштабування під великі обсяги потокових даних лишається недослідженим.

– пробіли. Недостатня увага приділяється інтерпретованості моделей у контексті кібербезпеки, адаптивному навчанню в умовах змін мережевого середовища, гнучким стратегіям оновлення моделі без повного перенавчання, а також комплексним гібридним рішенням, які можуть поєднувати ML/DL з

сигнатурними/експертними методами.

Аналіз сучасного стану проблеми свідчить, що хоча значна база знань вже існує, є достатньо наукового простору для розробки нових підходів, які б узгоджували продуктивність, адаптивність, пояснюваність та стійкість до загроз у реальному мережевому середовищі.

Мета дослідження полягає у розробці інтелектуальної системи кібербезпеки, здатної здійснювати моніторинг мережевого трафіку в реальному часі та автоматично виявляти вторгнення на основі алгоритмів машинного навчання, забезпечуючи підвищення точності класифікації загроз і зменшення кількості хибних спрацьовувань.

Для досягнення поставленої мети необхідно вирішити такі основні задачі дослідження:

– проаналізувати літературні джерела та визначити основні концепції мережевої безпеки;

– обґрунтувати вибір алгоритмів машинного навчання для задачі класифікації мережевого трафіку;

– розробити інтелектуальну систему моніторингу, що включає етапи підготовки даних, навчання моделі, її оцінювання та інтеграцію у процес реального аналізу мережевих потоків;

– провести експериментальну перевірку ефективності моделі, здійснити аналіз отриманих результатів.

Об’єкт дослідження – процес моніторингу та аналізу мережевого трафіку з метою виявлення і класифікації кіберзагроз у системах інформаційної безпеки.

Предмет дослідження – методи, алгоритми та програмні засоби застосування штучного інтелекту для автоматичного виявлення вторгнень і підвищення ефективності систем моніторингу мережевої активності.

Для реалізації поставленої мети та вирішення визначених завдань у роботі застосовано комплекс наукових методів:

– аналітичний метод – використано для узагальнення сучасних наукових підходів до побудови систем кіберзахисту, дослідження архітектур моделей

виявлення вторгнень і аналізу переваг машинного навчання у задачах класифікації мережевого трафіку;

– методи оброблення та аналізу даних – застосовано для попереднього очищення, нормалізації та перетворення вхідних параметрів трафіку, що формують вектор ознак моделі;

– методи статистичного аналізу – застосовано для кількісної оцінки ефективності моделі за показниками: Precision, Recall, F1, Accuracy, MicroAccuracy, MacroAccuracy, LogLoss і LogLossReduction, тощо;

– експериментальний метод – використано для перевірки працездатності створеної системи у процесі моделювання моніторингу мережевого трафіку.

Наукова новизна одержаних результатів. Наукова новизна полягає у розробленні інтелектуальної системи моніторингу мережевого трафіку, яка поєднує методи машинного навчання з механізмами автоматичного виявлення вторгнень. Запропоновано комплексний підхід до інтеграції числових і категорійних ознак трафіку в єдину модель класифікації, що забезпечує підвищену точність і стійкість до змін мережевого середовища.

Теоретичне значення роботи полягає в удосконаленні методологічних засад побудови систем кіберзахисту на основі штучного інтелекту, а також у формуванні підходу до оцінювання ефективності алгоритмів класифікації при аналізі мережевих загроз.

Практичне значення роботи полягає у створенні програмного модуля для автоматизованого виявлення кіберінцидентів у реальному часі, який може бути використаний для підвищення рівня безпеки інформаційно-комунікаційних систем різного типу. Отримані результати придатні для подальшої інтеграції у комплексні системи моніторингу кіберзагроз.

Галузь застосування. Результати дослідження можуть бути використані у підрозділах кібербезпеки підприємств, державних органів, фінансових структур і організацій критичної інфраструктури, а також у навчальному процесі підготовки фахівців з інформаційної безпеки та аналізу даних.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ПРОБЛЕМИ ТА ВИБІР НАПРЯМКІВ ДОСЛІДЖЕННЯ

1.1 Еволюція систем захисту інформації та концепція мережевої безпеки

Захист інформації в перші десятиліття розвитку комп'ютерних мереж базувався на трьох фундаментальних принципах – конфіденційності, цілісності та доступності. Конфіденційність забезпечує, щоб доступ до даних отримували лише уповноважені користувачі; цілісність гарантує правильність, достовірність та узгодженість інформації; доступність спрямована на забезпечення безперервного доступу до ресурсів для законних користувачів [11]. Ці принципи утворюють основу безпеки, яка залишається актуальною навіть у сучасних системах.

Перші мережеві засоби захисту були орієнтовані на обмеження доступу до ресурсів через контроль паролів, фізичне відокремлення та програмні фільтри. Зі зростанням масштабу мереж і появою комп'ютерних вірусів почали застосовувати міжмережеві екрани та антивіруси. У 1980-х роках Денінг представила систему виявлення вторгнень, що використовувала аудиторські записи та статистичні методи для виявлення аномалій. Ідея аналізу відхилень від нормальної поведінки стала ключовою для подальшого розвитку IDS [12].

З поширенням інтернету, хмарних обчислень та мобільних пристроїв традиційна концепція «твердої периферії», коли всі захищені ресурси знаходяться в межах корпоративної мережі, почала втрачати актуальність. Атаки стали багатоступеневими, а користувачі – розподіленими по всьому світу. Замість того щоб довіряти будь-якому пристрою за фактом його знаходження у внутрішній мережі, сучасні платформи керуються принципом Zero Trust: «нікому не довіряй, завжди перевіряй». Це означає постійний контроль доступу, багатофакторну автентифікацію, мікросегментацію та мінімізацію привілеїв. Аналітики IBM відзначають, що моделі Zero Trust можуть бути орієнтовані на ідентичність, мережу, дані, застосунки або пристрої, але у всіх випадках принципи полягають у постійному нагляді, застосуванні політики найменших привілеїв та мікросегментації [13]. Ця концепція зменшує ризик lateral movement –

горизонтального переміщення зловмисника між сегментами мережі.

Системи виявлення вторгнень (IDS) відіграють фундаментальну роль у сучасній архітектурі кібербезпеки, забезпечуючи безперервний моніторинг подій у мережевому середовищі та аналітичне оцінювання активності користувачів і процесів з метою своєчасного виявлення деструктивних або потенційно небезпечних дій. Класифікація таких систем ґрунтується на сукупності ознак, які визначають їхню структурну організацію, принципи функціонування, методи ідентифікації загроз і способи реагування на виявлені інциденти. Такий підхід дозволяє систематизувати IDS за функціональними параметрами, визначити їх придатність до конкретних умов експлуатації, оцінити переваги й обмеження кожного класу, а також розробити комбіновані стратегії інтеграції для підвищення рівня стійкості мережевої інфраструктури до кібератак.

Одним із базових критеріїв класифікації виступає середовище розгортання. Мережеві IDS орієнтовані на аналіз трафіку у режимі реального часу, здійснюючи перехоплення та дослідження пакетів даних, що проходять через комунікаційні вузли [14]. Це дає змогу виявляти аномальні шаблони поведінки, зокрема спроби сканування портів або ініціацію несанкціонованих з’єднань. Проте ефективність таких систем знижується при обробленні зашифрованого трафіку, що обмежує їх застосування у середовищах із підвищеними вимогами до конфіденційності. Хостові системи, у свою чергу, аналізують події на рівні конкретного пристрою чи сервера, фіксуючи зміни у системних файлах, діях користувачів і процесах. Гібридні рішення поєднують обидва підходи, забезпечуючи комплексний аналіз поведінки як на рівні мережі, так і окремих вузлів, що підвищує точність виявлення складних багаторівневих атак.

Критичною ознакою класифікації IDS є також метод виявлення загроз. Сигнатурний підхід ґрунтується на пошуку збігів між вхідними даними та базою відомих шаблонів атак, забезпечуючи високу точність розпізнавання вже задокументованих загроз [15]. Водночас він демонструє низьку ефективність при зустрічі з новими, невідомими сценаріями атак. Поведінкові або статистичні системи використовують моделі машинного навчання чи аналіз відхилень для

виявлення нетипової активності. Завдяки цьому вони здатні виявляти нові типи вторгнень, однак схильні до підвищеного рівня хибних спрацювань, особливо у складних динамічних середовищах [16]. Найсучаснішим напрямом розвитку вважаються гібридні IDS, які інтегрують сигнатурний і поведінковий підходи, створюючи багаторівневу систему аналізу загроз, здатну до самонавчання та адаптації у процесі експлуатації.

Для ілюстрації ключових відмінностей між зазначеними методами у табл. 1.1 наведено їх порівняльну характеристику. Наведені дані демонструють відмінності у принципах роботи, перевагах і недоліках кожного підходу.

Таблиця 1.1

Порівняння основних методів виявлення вторгнень

Метод виявлення	Принцип роботи	Переваги	Недоліки
Сигнатурний	Порівняння вхідних даних з базою відомих шаблонів	Висока точність для відомих атак, низький рівень хибних спрацювань	Неможливість виявлення нових типів атак
Поведінковий	Аналіз відхилень від звичайної поведінки системи	Може виявляти нові, невідомі атаки	Велика кількість хибних сповіщень
Гібридний	Комбінація сигнатурного та поведінкового підходів	Оптимальне поєднання точності та адаптивності	Підвищені вимоги до обчислювальних ресурсів

Сучасні системи виявлення вторгнень поділяють також за принципом навчання, який використовується для побудови моделі поведінки мережі або окремих її компонентів. У системах із навчанням під наглядом (supervised learning) застосовується маркований набір даних, де кожен запис попередньо класифіковано як нормальна активність або атака. Такий підхід властивий класичним алгоритмам машинного навчання, зокрема Random Forest, Support Vector Machine (SVM) та Gradient Boosting, які формують моделі на основі узагальнення закономірностей у відомих вибірках [17]. На відміну від цього, системи з навчанням без нагляду

(unsupervised learning) функціонують без попереднього маркування даних, самостійно виявляючи схожі кластери поведінки та відхилення, що дає змогу виявляти раніше невідомі аномалії. До цієї категорії належать алгоритми K-Means, DBSCAN та Autoencoders, які ефективно ідентифікують нетипові взаємозв'язки у великих масивах трафіку [18].

У новітніх розробках IDS усе ширше використовуються моделі з підкріпленням (reinforcement learning), орієнтовані на поступове вдосконалення своїх дій завдяки отриманому досвіду [19]. Такі системи формують адаптивні політики реагування, змінюючи стратегію виявлення та протидії залежно від результатів попередніх рішень. Це дозволяє підвищити стійкість до змінних типів загроз і реалізувати динамічний контроль безпеки у реальному часі.

Важливою ознакою класифікації виступає тип реакції системи. Пасивні IDS зосереджені на детекції загроз і передаванні повідомлень адміністраторам, не втручаючись безпосередньо у трафік. На відміну від них, активні системи (IPS – Intrusion Prevention Systems) здійснюють автоматичне блокування атакуючих з'єднань, ізолюють підозрілі вузли та можуть змінювати політику доступу для локалізації інциденту [20]. Такий підхід забезпечує скорочення часу реагування та зниження ризику масштабування атаки.

Додатковим параметром класифікації є архітектурна організація системи. Централізовані IDS акумулюють дані в єдиному центрі аналітики, що спрощує управління та контроль, проте створює потенційний ризик перевантаження або втрати функціональності у випадку збою центрального вузла. Розподілені рішення базуються на кількох автономних модулях, які функціонують у різних сегментах мережі, забезпечуючи обмін інформацією та підвищення масштабованості. Хмарні системи орієнтовані на аналітичну обробку потоків даних у віддаленому середовищі, використовуючи високопродуктивні обчислювальні ресурси для реалізації складних алгоритмів штучного інтелекту.

Узагальнені класифікаційні ознаки IDS наведено в табл. 1.2, яка відображає системну структуру підходів до виявлення та реагування на кіберзагрози. Представлені дані окреслюють логіку еволюції від традиційних централізованих

систем до адаптивних інтелектуальних платформ, здатних до автономного аналізу, прогнозування й самонавчання в умовах змінного інформаційного середовища.

Таблиця 1.2

Узагальнення класифікаційних ознак систем виявлення вторгнень

Ознака класифікації	Тип системи	Основна характеристика
Середовище розгортання	Мережева, хостова, гібридна	Аналіз трафіку або локальних подій
Метод виявлення	Сигнатурний, поведінковий, гібридний	Виявлення шаблонів чи поведінкових відхилень
Тип навчання	З учителем, без учителя, з підкріпленням	Формування моделей поведінки на основі даних
Тип реакції	Пасивна, активна	Повідомлення чи автоматичне блокування
Архітектура	Централізована, розподілена, хмарна	Спосіб організації обчислень і зберігання даних

Таким чином, класифікація IDS демонструє еволюцію від статичних сигнатурних систем до динамічних, інтелектуальних рішень, здатних самонавчатися і пристосовуватися до змін середовища. Поєднання різних типів IDS у межах однієї архітектури забезпечує комплексний підхід до моніторингу мережі, що є основою ефективної системи кібербезпеки нового покоління.

1.2 Інтелектуальні методи виявлення вторгнень

Розвиток інформаційно-комунікаційних систем супроводжується безпрецедентним зростанням обсягів мережевого трафіку, у структурі якого дедалі частіше приховується зловмисна активність. Традиційні методи захисту, засновані на сигнатурному аналізі або жорстко заданих правилах, виявилися недостатньо ефективними в умовах швидкої еволюції кіберзагроз. Саме тому на перший план виходять інтелектуальні IDS, побудовані на основі алгоритмів машинного та глибинного навчання, здатних автоматично виявляти закономірності, притаманні поведінці атак.

Інтелектуальні IDS не обмежуються пошуком відомих сигнатур, а формують статистичну чи нейромережеву модель «нормальної поведінки» системи. Завдяки цьому можливе виявлення аномалій, які не мають попередньо відомих ознак. Такий підхід забезпечує адаптивність і здатність реагувати на нові, раніше невідомі загрози, що істотно підвищує загальний рівень кіберзахисту.

Основу побудови інтелектуальних систем становлять кілька ключових етапів оброблення мережевого трафіку:

– збір і попередня обробка даних, включно з очищенням, нормалізацією та кодуванням ознак;

– виділення релевантних характеристик потоків (feature extraction) для зменшення розмірності;

– навчання моделей класифікації або виявлення аномалій;

– оцінка ефективності за допомогою метрик точності, повноти, F1-міри та AUC;

– інтерпретація результатів і автоматизація реагування на виявлені інциденти.

Ці етапи реалізуються різними методологічними підходами, що визначають типи та архітектуру інтелектуальних систем. Загалом сучасні методи IDS можна поділити на чотири групи:

Методи машинного навчання – базуються на використанні алгоритмів класифікації та регресії, таких як Decision Tree, Random Forest, SVM, або Logistic Regression [21]. Ці підходи ефективні при наявності маркованих наборів даних.

Методи глибинного навчання – передбачають використання нейронних мереж із багатьма шарами (CNN, LSTM, Autoencoder), здатних самостійно виявляти складні нелінійні залежності між параметрами трафіку [22].

Ансамблеві методи – поєднують результати кількох алгоритмів для підвищення стабільності та точності класифікації, застосовуючи техніки boosting, bagging або stacking.

Гібридні системи (Hybrid IDS) – інтегрують ШІ-моделі з класичними сигнатурними або експертними підходами, забезпечуючи баланс між швидкодією

та адаптивністю.

Порівняльну характеристику традиційних і інтелектуальних методів виявлення вторгнень наведено в табл. 1.3, яка відображає принципові відмінності між підходами за основними параметрами – джерелом знань, здатністю до навчання, адаптивністю, точністю та ефективністю в умовах нових типів атак.

Таблиця 1.3

Порівняння традиційних і інтелектуальних методів виявлення вторгнень

Ознака порівняння	Традиційні IDS	Інтелектуальні IDS
Джерело знань	База сигнатур і наперед визначених правил	Навчальні дані та адаптивна модель поведінки
Реакція на нові атаки	Обмежена, лише після оновлення сигнатур	Може виявляти нові або невідомі типи загроз
Гнучкість та масштабованість	Залежить від експертних правил	Висока, модель самонавчається на нових даних
Необхідність ручного налаштування	Висока, потребує участі аналітика	Мінімальна, автоматизоване навчання
Точність класифікації	Висока для відомих атак, низька для нових	Висока як для відомих, так і для нових атак
Обчислювальні вимоги	Невеликі	Залежать від складності моделі (ML/DL)
Підтримка роботи в реальному часі	Обмежена	Реалізується за наявності оптимізованих моделей
Приклади систем	Snort, Suricata, Bro/Zeek	AI-IDS, DeepIDS, HybridML-IDS

Інтелектуальні методи виявлення вторгнень базуються на математичних моделях, здатних навчатися на прикладах попередніх кіберінцидентів і формувати узагальнене уявлення про поведінку системи. На відміну від статичних сигнатурних підходів, алгоритми машинного та глибинного навчання враховують статистичні взаємозв’язки між числовими та категорійними параметрами трафіку, що забезпечує динамічне пристосування моделі до нових умов мережевого середовища.

Класичні алгоритми машинного навчання широко використовуються у

системах IDS завдяки здатності виконувати класифікацію, регресію та кластеризацію. Найбільш поширеними є такі:

– Decision Tree – алгоритм, що будує ієрархію рішень на основі інформаційного приросту або ентропії [23]. Його перевагою є інтерпретованість, а недоліком – ризик перенавчання на великих наборах ознак;

– Random Forest – ансамбль рішень, що усереднює результати багатьох дерев [24]. Завдяки цьому метод демонструє високу стійкість до шумів та переобчислення, зберігаючи стабільну точність навіть при значній розмірності даних.

– Support Vector Machine – алгоритм, що будує гіперплощину, яка розділяє класи у багатовимірному просторі ознак [25]. Його ефективність висока для невеликих та середніх за обсягом наборів даних із чіткими кордонами між класами.

– Logistic Regression – модель для двокласової класифікації, що добре описує лінійно відокремлювані дані, проте має обмежену здатність до узагальнення при складних нелінійних залежностях.

– Stochastic Dual Coordinate Ascent (SDCA) – алгоритм оптимізації, що використовується в середовищі ML.NET. Його ключовою перевагою є ефективність при роботі з великими наборами даних і підтримка багатокласової класифікації. Завдяки швидкій збіжності SDCA забезпечує високу точність виявлення загроз при помірному споживанні ресурсів, що робить його оптимальним вибором для побудови системи реального часу.

Методи глибинного навчання формують підґрунтя сучасних систем IDS, здатних автоматично виділяти суттєві ознаки з великих обсягів даних. Вони особливо ефективні при аналізі складних мережевих залежностей і багатовимірних структур даних. Найбільш поширеними методами глибинного навчання є:

– Convolutional Neural Network (CNN) – модель, що виконує просторову фільтрацію даних і добре виявляє локальні закономірності [26]. У контексті IDS використовується для аналізу структурних шаблонів трафіку та виявлення повторюваних патернів атак;

– Long Short-Term Memory (LSTM) – різновид рекурентних мереж, здатний

враховувати часову послідовність подій. LSTM-моделі ефективні при аналізі потокового трафіку, де характер поведінки змінюється у часі [27];

– Autoencoder – нейронна мережа, що навчається відтворювати вхідні дані, зберігаючи лише їхню найсуттєвішу структуру [28]. Аномалії, що суттєво відрізняються від “норми”, мають підвищену похибку реконструкції й інтерпретуються як потенційні атаки.

Кожен із зазначених алгоритмів має власну область оптимального застосування залежно від типу вхідних даних, обчислювальних ресурсів і вимог до інтерпретованості. Узагальнену характеристику основних методів машинного та глибинного навчання наведено у табл. 1.4, де систематизовано їхні властивості, переваги та обмеження.

Таблиця 1.4

Порівняння традиційних і інтелектуальних методів виявлення вторгнень

Алгоритм	Основна ідея	Переваги	Недоліки
1	2	3	4
Decision Tree	Ієрархічна побудова правил на основі інформаційного приросту	Інтерпретованість, простота реалізації	Схильність до переобчислення
Random Forest	Комбінація багатьох дерев рішень	Висока точність, стійкість до шуму	Вища обчислювальна складність
Support Vector Machine	Побудова оптимальної гіперплощини між класами	Висока точність для чітких меж	Погана масштабованість на великих даних
Logistic Regression	Лінійна модель для класифікації	Швидкість, інтерпретованість	Обмежена нелінійна здатність
Stochastic Dual Coordinate Ascent	Стохастична оптимізація подвійної задачі	Висока швидкість і стабільність збіжності	Менша ефективність при сильно нелінійних залежностях
CNN	Просторове згортання даних	Автоматичне виділення ознак	Високі обчислювальні витрати

Продовження таблиці 1.4.

1	2	3	4
LSTM	Запам’ятовування часових залежностей	Ефективність при потоковому трафіку	Висока складність навчання
Autoencoder	Відновлення структури даних для пошуку аномалій	Виявлення невідомих атак	Залежність від налаштувань і вибору ознак

Поєднання класичних методів машинного навчання (зокрема SDCA) з елементами глибинного навчання дозволяє створити збалансовану модель, яка одночасно забезпечує точність класифікації, швидкість реагування та можливість інтеграції у реальні мережеві системи. Особливістю обраного підходу є те, що він орієнтований на адаптивне навчання моделі, здатної обробляти потоки даних у реальному часі, з подальшою можливістю розширення до гібридної архітектури.

Подальший розвиток інтелектуальних систем виявлення вторгнень відбувається в напрямі створення ансамблевих та гібридних моделей, що поєднують різні підходи з метою підвищення точності класифікації, стійкості до шуму та здатності виявляти складні багаторівневі загрози. Такі системи забезпечують гнучке балансування між обчислювальною ефективністю та глибиною аналізу, зберігаючи можливість роботи у режимі реального часу.

Ансамблеве навчання (Ensemble Learning) полягає у використанні декількох моделей-класифікаторів, результати яких комбінуються для отримання більш узгодженого рішення [29]. Основна ідея полягає в тому, що помилки окремих моделей компенсуються узагальненим результатом, що забезпечує підвищення загальної точності системи.

Найбільш поширеними ансамблевими методами є:

– Bagging (Bootstrap Aggregating) – створення кількох моделей на різних підмножинах даних із подальшим усередненням результатів. Класичним прикладом є алгоритм Random Forest;

– Boosting (AdaBoost, Gradient Boosting, XGBoost, LightGBM) – послідовне побудування моделей, де кожна наступна коригує помилки попередньої. Такі

підходи демонструють високу точність у багатокласових задачах виявлення атак;

– Stacking (Stacked Generalization) – використання метамоделі, яка поєднує виходи базових моделей (наприклад, SVM, RF, LR), створюючи оптимальне рішення шляхом їхньої узгодженої інтеграції.

Згідно з результатами дослідження [30], ансамблеві підходи показують підвищення середньої точності класифікації атак на 10–15 % у порівнянні з поодинокими моделями машинного навчання. Вони особливо ефективні у випадках, коли мережевий трафік характеризується великою варіативністю та неоднорідністю типів атак.

Гібридні IDS поєднують традиційні методи (сигнатурні, статистичні, евристичні) з інтелектуальними моделями машинного або глибинного навчання. Такий підхід дає змогу одночасно зберегти переваги швидкості виявлення відомих атак і гнучкості щодо нових або невідомих загроз.

Типовий приклад гібридної системи включає дві стадії аналізу:

– фільтраційна (детермінована) – використання сигнатурного або правило-орієнтованого механізму для відсіву відомих типів атак;

– аналітична (інтелектуальна) – застосування моделі машинного навчання для детального аналізу залишкового трафіку, що може містити нові або модифіковані атаки.

Серед найвідоміших реалізацій – Hybrid-ML IDS, де використовуються нейронні мережі в поєднанні з Random Forest або XGBoost для підвищення узагальнювальної здатності моделі. Такі системи відзначаються високою точністю (до 98 %) при прийнятному рівні затримок, що підтверджує їх ефективність у промислових мережах і системах критичної інфраструктури [31].

Попри значні успіхи, інтелектуальні IDS стикаються з низкою викликів, які визначають подальші напрями досліджень:

– дисбаланс класів у наборах даних, коли кількість записів із нормальним трафіком значно перевищує кількість атак, що ускладнює навчання моделі;

– adversarial-атаки, спрямовані на введення моделі в оману шляхом навмисної модифікації вхідних параметрів;

– інтерпретованість рішень – необхідність пояснення, чому система класифікує певний пакет як шкідливий. У цьому контексті інтеграція методів Explainable AI (XAI), таких як LIME або SHAP, набуває особливої актуальності;

– проблема масштабування – потреба у швидкій обробці великих потоків даних у реальному часі без втрати точності класифікації.

Враховуючи наведені фактори, перспективним напрямом є розроблення гібридних інтелектуальних систем, які поєднують переваги класичних сигнатурних методів із самонавчальними моделями машинного та глибинного навчання, забезпечуючи баланс між продуктивністю, адаптивністю та пояснюваністю.

Порівняльний аналіз основних груп інтелектуальних методів виявлення вторгнень наведено у табл. 1.5, яка систематизує їхні характеристики за ключовими критеріями: точність, швидкодія, гнучкість, потреба у даних та інтерпретованість.

Таблиця 1.5

Порівняльний аналіз сучасних інтелектуальних методів IDS

Критерій	Класичні ML-методи	Глибинне навчання	Ансамблеві методи	Гібридні системи
Швидкодія	Висока	Середня (потребує GPU)	Середня	Висока при оптимізованій інтеграції
Гнучкість і адаптивність	Середня	Висока	Висока	Дуже висока
Потреба у великих даних	Помірна	Значна	Середня	Середня
Інтерпретованість	Висока	Низька	Середня	Середня/висока (з XAI)
Стійкість до шуму	Середня	Висока	Висока	Висока
Рівень застосування	Малі та середні мережі	Високонавантажені системи	Корпоративні IDS	Критична інфраструктура

Як видно з табл. 1.5, найвищі показники продуктивності досягаються саме при поєднанні кількох інтелектуальних підходів. Гібридні системи забезпечують не

лише підвищення точності, а й покращення адаптивності до змінного середовища, водночас зберігаючи можливість пояснення рішень за допомогою XAI-модулів.

Узагальнюючи викладене, можна стверджувати, що інтелектуальні методи виявлення вторгнень формують нову парадигму кіберзахисту, де основна увага приділяється не стільки пасивному виявленню атак, скільки активному прогнозуванню загроз на основі навчання системи. Це забезпечує перехід від реактивних до превентивних стратегій безпеки, що є визначальним чинником підвищення стійкості інформаційних інфраструктур.

1.3 Огляд наукових джерел з проблематики виявлення та класифікації кіберзагроз

Використання алгоритмів машинного та глибинного навчання у процесах виявлення і класифікації кіберзагроз стало однією з провідних тенденцій розвитку інформаційних систем безпеки. Наукові публікації 2020–2025 рр. демонструють системний перехід від реактивних сигнатурних механізмів до інтелектуальних систем виявлення вторгнень, що здатні адаптивно реагувати на динаміку мережевих подій. Згідно з результатами аналізу, проведеного у статті [32], наукова активність у цьому напрямі зросла більш ніж утричі порівняно з періодом 2015–2019 рр. Дослідники відзначають, що ключовим фактором такої динаміки є розвиток методів глибинного навчання та можливість автоматичного аналізу складних структур даних у реальному часі.

У межах огляду автори класифікували основні напрями використання ШІ в кібербезпеці, серед яких визначено:

– виявлення вторгнень – побудова моделей класифікації мережевого трафіку;

– аналіз шкідливого програмного забезпечення– розпізнавання шкідливих виконуваних файлів і поведінкових патернів;

– поведінковий аналіз користувачів – ідентифікація аномальної активності;

– розвідка загроз – прогнозування майбутніх атак на основі минулих тенденцій;

– захист хмарних і IoT-систем – контроль доступу, моніторинг і моделювання

ризиків у розподілених середовищах.

Ці напрями узагальнено у рис. 1.1, адаптованому за даними роботи [32], який демонструє ключові області застосування технологій ШІ та МН у кібербезпеці та відображає їхню взаємодію в єдиному контексті інформаційного захисту.

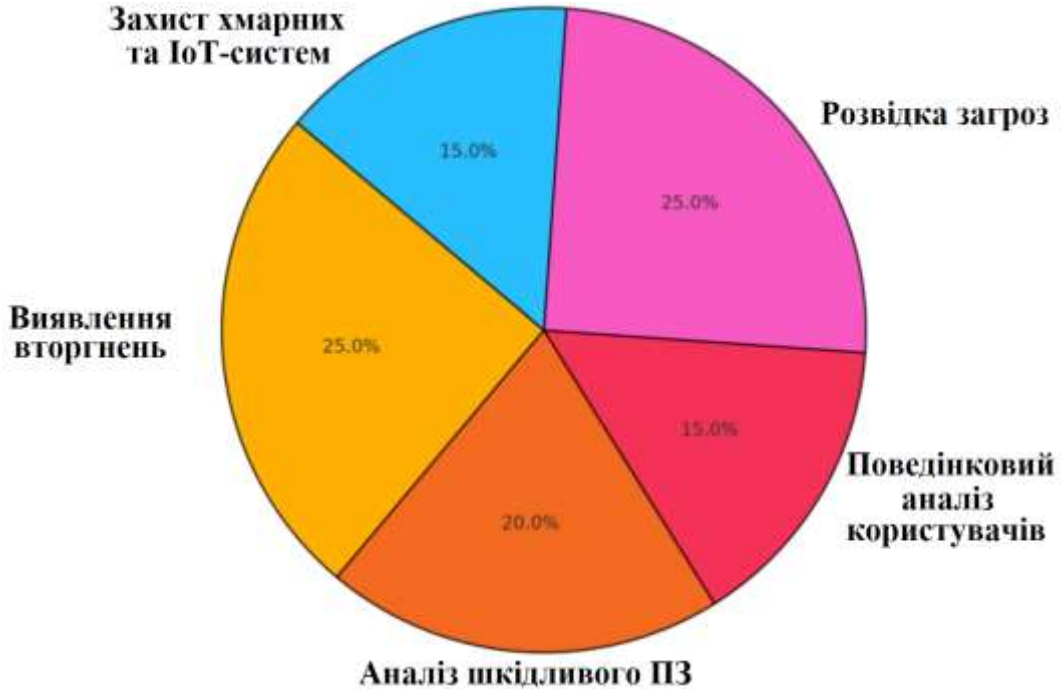

Рис. 1.1 Області застосування ШІ та машинного навчання у кібербезпеці

Як свідчить систематизація, головним пріоритетом досліджень у 2023–2025 рр. є створення інтелектуальних систем виявлення та класифікації кіберзагроз, здатних працювати на великих потоках мережевого трафіку. У роботі [33] зазначено, що алгоритми Random Forest, Gradient Boosting, Decision Tree, CNN та LSTM демонструють середню точність у межах 92–99 % при тестуванні на наборах NSL-KDD, CICIDS2017, UNSW-NB15.

Зокрема, згідно з даними [34], моделі LSTM та CNN перевищують класичні методи (SVM, RF) за точністю на 3–5 %, що пов’язано з їхньою здатністю виявляти приховані часові закономірності у трафіку. Автори також підкреслюють, що інтеграція таких моделей у корпоративні мережі забезпечує не лише підвищення точності класифікації атак, а й скорочення часу реагування в середньому на 37 %.

Водночас у звіті [35] зазначено, що понад 70 % організацій, які впровадили ШІ-системи кіберзахисту, стикаються з проблемою відсутності пояснюваності моделей, що ускладнює інтерпретацію рішень та знижує рівень довіри до

автоматизованих рішень у критичних інфраструктурах. Це стимулює розвиток напрямку Explainable AI, спрямованого на створення прозорих механізмів інтерпретації результатів.

У сучасних дослідженнях з кібербезпеки ключова увага зосереджується не лише на розробленні архітектур систем виявлення вторгнень, а й на виборі оптимальних методів машинного навчання, що визначають якість класифікації атак і швидкість реагування системи. Рівень застосування конкретних методів штучного інтелекту та машинного навчання у сфері кіберзахисту істотно відрізняється залежно від типу завдання, обсягу даних і потреб у пояснюваності моделі.

Відповідно до узагальнених даних, наведених у [36], переважна більшість систем (понад 60 %) базується на навчанні з учителем (supervised learning). Цей підхід використовує марковані набори даних, де кожен зразок містить інформацію про належність до певного класу («нормальний трафік» або «атака»). Алгоритми, такі як Random Forest, SVM та Logistic Regression, демонструють найвищу точність на класичних датасетах (NSL-KDD, CICIDS2017, UNSW-NB15), досягаючи 90–99 % при належній підготовці даних.

Водночас методи без учителя (unsupervised learning), що охоплюють кластеризацію (K-Means, DBSCAN) та автоенкодери, використовуються у близько 25 % наявних рішень. Їхня головна перевага полягає у здатності виявляти невідомі типи атак, однак складність інтерпретації результатів часто обмежує їх практичне застосування.

Менш поширеним є використання навчання з підкріпленням, яке становить близько 15 % досліджуваних випадків і застосовується переважно у системах активного реагування, де модель вчиться оптимізувати дії на основі нагороди або штрафу. Цей підхід успішно використовується у сценаріях автоматичного керування міжмережевими екранами та адаптивного налаштування IDS-параметрів.

Найменшу частку мають підходи, засновані на автоматизації та адаптивній відповіді – приблизно 10–15 %, що зумовлено потребою в значних обчислювальних ресурсах і складністю впровадження у корпоративні мережі. Проте саме ці напрями

активно розвиваються у дослідженнях останніх років, оскільки дозволяють системам не лише виявляти, але й самостійно реагувати на виявлені інциденти, зменшуючи участь людини-аналітика.

Узагальнення цієї статистики наведено на рис. 1.2, який відображає частку застосування ключових технік AI/ML у галузі кібербезпеки за даними 2023–2024 років.

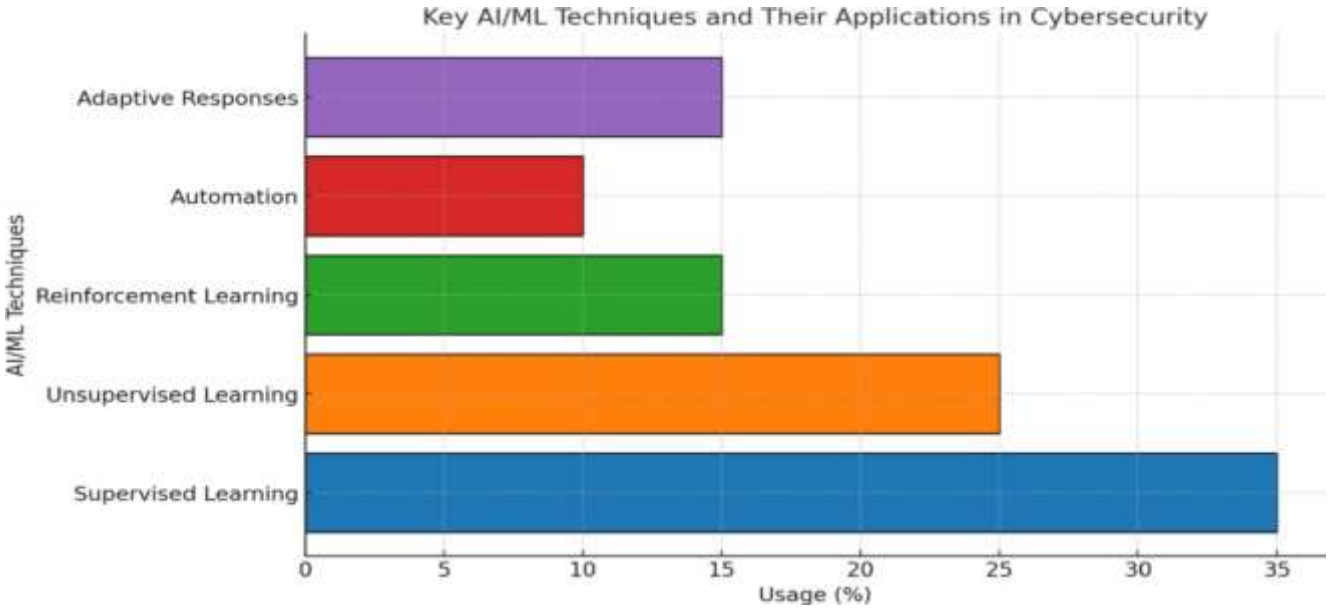

Рис. 1.2 Основні методи AI/ML та сфери їх застосування у кібербезпеці

Аналіз наведених джерел свідчить, що інтеграція різних AI-технологій забезпечує синергетичний ефект: комбінація supervised та unsupervised-підходів дозволяє підвищити точність до 97–98 % і водночас знизити рівень хибних спрацьовувань. Це визначає головну тенденцію розвитку – перехід до гібридних систем виявлення вторгнень, які поєднують класифікаційні моделі з алгоритмами прогнозування та адаптивного реагування.

Українські дослідники також роблять вагомий внесок у розвиток інтелектуальних систем кіберзахисту, спрямованих на виявлення та протидію складним загрозам у критичній інфраструктурі. Зокрема, у роботі Морозова, Вакалюк, Єфіменка та співавторів представлено підхід, який поєднує технології honeypot-систем і кіберобману (cyber deception) для підвищення точності виявлення атак у промислових мережах [37]. Автори підкреслюють, що створення інтегрованого середовища симуляції атак дозволяє не лише виявляти зовнішні

вторгнення, а й аналізувати поведінку зловмисника під час проникнення, що формує основу для навчання моделей штучного інтелекту на реалістичних сценаріях загроз.

Дослідження показало, що використання кіберобману у поєднанні з аналітичними алгоритмами машинного навчання сприяє зниженню кількості хибних спрацьовувань систем виявлення на близько 25–30 %, водночас підвищуючи ефективність раннього розпізнавання складних і багаторівневих атак на критичні об’єкти інфраструктури. Це доводить, що інтеграція класичних технологій мережевого моніторингу з елементами інтелектуального аналізу є одним із найперспективніших напрямів розвитку сучасних систем кібербезпеки.

Узагальнення результатів українських та зарубіжних досліджень свідчить, що найвищі показники точності – до 97–99 % – демонструють гібридні архітектури, які комбінують навчання з учителем та без учителя. Такі системи здатні автоматично формувати профілі атак, адаптуватися до змін у мережевому середовищі та забезпечувати стійкість до невідомих загроз.

Разом із тим, ключовими викликами залишаються:

– необхідність формування великих, збалансованих і репрезентативних навчальних вибірок;

– складність інтерпретації рішень у моделях глибокого навчання;

– забезпечення масштабованості систем у розподілених середовищах IoT, Edge та Cloud.

Отже, сучасні тенденції розвитку українських і міжнародних досліджень підтверджують, що створення адаптивних, пояснюваних і енергоефективних систем виявлення кіберзагроз є стратегічним напрямом еволюції інтелектуальних технологій безпеки, здатних не лише ідентифікувати, а й прогнозувати загрози у динамічних цифрових екосистемах.

1.4 Постановка наукової задачі та визначення напрямів подальших досліджень

Аналіз сучасних підходів до виявлення кіберзагроз засвідчив, що, попри

активне впровадження алгоритмів машинного та глибинного навчання, існуючі системи не повною мірою відповідають вимогам адаптивності та надійності. Основні труднощі пов’язані з обробленням великих обсягів трафіку в умовах постійної зміни мережевого середовища, автоматичним оновленням моделей без участі фахівця, а також необхідністю пояснення результатів прогнозування для забезпечення довіри до системи. Традиційні системи виявлення вторгнень зосереджені переважно на ідентифікації відомих сигнатур атак, тоді як сучасна кібербезпека вимагає інтелектуальних рішень, здатних аналізувати поведінкові закономірності користувачів і розпізнавати нетипові події, що можуть свідчити про нові або приховані загрози.

У межах цього дослідження поставлено завдання створити інтелектуальну систему кіберзахисту, яка забезпечує моніторинг і виявлення вторгнень у мережевому трафіку засобами машинного навчання. Основна ідея полягає у побудові адаптивної моделі, що здатна класифікувати події мережевого середовища, аналізувати їх у реальному часі та автоматично коригувати власні параметри залежно від змін у структурі трафіку.

Для досягнення цієї мети передбачено реалізацію низки етапів, які визначають логіку подальшого дослідження:

– обґрунтування вибору алгоритму машинного навчання, аналіз придатності до різних типів атак і формування критеріїв порівняння;

– розроблення методики підготовки даних, що охоплює очищення, нормалізацію та балансування навчальних і тестових вибірок;

– побудову архітектури аналітичного модуля системи моніторингу, який поєднує моделі класифікації, поведінкового аналізу та оцінки ризику;

– створення програмного прототипу для проведення експериментів, тестування точності, повноти, F1-міри та AUC-ROC.

Виконання цих завдань забезпечить наукове й практичне підґрунтя для побудови цілісної, адаптивної та масштабованої системи моніторингу, здатної ефективно протидіяти сучасним кібератакам у динамічному мережевому середовищі.

Висновки до першого розділу

У рамках даного розділу здійснено аналіз сучасних підходів до побудови систем виявлення та класифікації кіберзагроз, що дозволило сформувати цілісне уявлення про еволюцію засобів захисту інформації та визначити основні напрями подальших досліджень. Розглянуто розвиток концепції мережевої безпеки та узагальнено класифікаційні ознаки систем виявлення вторгнень за середовищем розгортання, методом виявлення, типом навчання, реакцією системи та архітектурою. Порівняння сигнатурних, поведінкових і гібридних підходів показало, що саме гібридні IDS поєднують переваги точності, адаптивності та здатності реагувати на невідомі типи атак, що робить їх базою для побудови сучасних інтелектуальних систем кіберзахисту.

Проаналізовано інтелектуальні методи виявлення вторгнень, які використовують алгоритми машинного навчання та глибинних нейронних мереж. Порівняльний огляд таких методів, як Decision Tree, Random Forest, Support Vector Machine, Logistic Regression, CNN, LSTM та Autoencoder, дозволив визначити їхні сильні сторони та обмеження у контексті застосування для аналізу мережевого трафіку. На основі отриманих результатів встановлено, що ансамблеві та гібридні архітектури демонструють найкращі показники точності й стійкості до шуму, тоді як глибинні нейронні мережі забезпечують автоматичне виділення ознак, необхідне для виявлення складних багатовимірних атак.

Огляд сучасних наукових джерел засвідчив активний розвиток напрямів інтеграції штучного інтелекту у сферу кібербезпеки, зокрема у завданнях виявлення вторгнень, аналізу шкідливого програмного забезпечення, поведінкової аналітики користувачів, моніторингу IoT-інфраструктур та захисту хмарних середовищ. На основі вивчених праць українських і зарубіжних авторів встановлено, що найвищі результати забезпечують гібридні системи, які поєднують навчання з учителем і без учителя, здатні до самооновлення та адаптації до нових кіберзагроз.

Здійснено формалізацію наукової задачі, спрямованої на створення інтелектуальної системи кіберзахисту з використанням алгоритмів машинного

навчання для виявлення вторгнень у мережевому трафіку. Сформульовано ключові завдання дослідження, що передбачають вибір ефективних алгоритмів класифікації, розроблення методики підготовки даних, побудову архітектури аналітичного модуля та створення програмного прототипу для експериментального дослідження.

Отримані результати створюють наукове підґрунтя для розроблення теоретичних і практичних методів у подальшому дослідженні. Вони дозволяють перейти до побудови математичних моделей, вибору оптимальних алгоритмів машинного навчання, формування навчальних і тестових вибірок, а також розроблення архітектури інтелектуальної системи моніторингу мережевого трафіку, що буде реалізовано у наступному розділі роботи.

РОЗДІЛ 2. ТЕОРЕТИЧНІ ТА ЕКСПЕРИМЕНТАЛЬНІ МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Вибір та обґрунтування алгоритмів машинного навчання для виявлення атак

Ефективність інтелектуальної системи кіберзахисту безпосередньо залежить від коректного вибору алгоритмів машинного навчання, які забезпечують високу точність класифікації мережевих подій і здатність адаптуватися до динаміки кіберзагроз. На сучасному етапі розвитку систем виявлення вторгнень пріоритетними вважаються алгоритми, що поєднують здатність до узагальнення, стійкість до шуму та інтерпретованість результатів. Це обумовлено тим, що у мережевих даних часто присутні аномалії, нерівномірні класи та залежності між атрибутами, що ускладнює завдання класифікації.

Алгоритм Бройдена–Флетчера–Голдфарба–Шанно (БФГШ) належить до класу ітераційних методів оптимізації, що використовуються для мінімізації багатовимірних нелінійних функцій без обмежень [38]. Його суть полягає в наближенні оберненої матриці Гессе (другої похідної цільової функції) без необхідності її прямого обчислення. Це дозволяє істотно зменшити обчислювальні витрати та забезпечити швидку збіжність навіть для задач великої розмірності. Алгоритм застосовується у різних галузях – від чисельного аналізу до машинного навчання, зокрема при навчанні штучних нейронних мереж, де він оптимізує вагові коефіцієнти для зменшення похибки моделі.

БФГШ є однією з найпоширеніших модифікацій методу Ньютона, оскільки він поєднує високу точність з ефективністю реалізації. Його ключова перевага – у використанні наближеної матриці Гессе, яка оновлюється на кожній ітерації на основі градієнтів функції помилки, що забезпечує баланс між швидкістю збіжності та стабільністю обчислень. Такий підхід особливо корисний у задачах машинного навчання, де поверхня функції втрат часто є складною, з великою кількістю локальних мінімумів.

На рис. 2.1 наведено структурну схему роботи нейронної мережі, яка реалізує

процес оптимізації з використанням алгоритму Бройдена–Флетчера–Голдфарба–Шанно. Мережа отримує на вхід вектор ознак $x_1,x_2,...,x_n$, що зважуються відповідними коефіцієнтами $w_{i1},w_{i2},...,w_{in}$. Далі обчислюється зважена сума сигналів (функція суми), до якої додається зсув b. Отримане значення net_j передається на активаційну функцію, яка визначає вихідний сигнал O_j. На етапі зворотного поширення помилки алгоритм БФГШ коригує вагові коефіцієнти, мінімізуючи функцію втрат і забезпечуючи поступове навчання моделі.

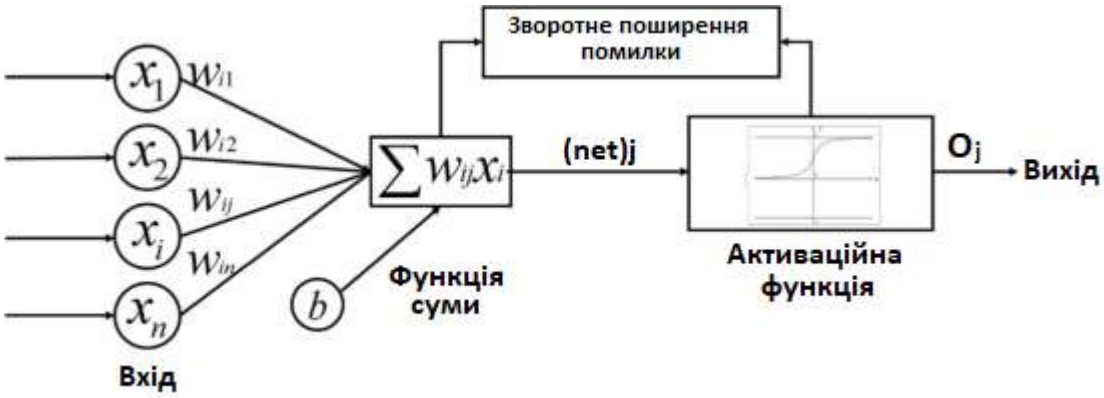

Рис. 2.1 Схема алгоритму «Бройдена-Флетчера-Голдфарба-Шанно» [39]

Застосування алгоритму БФГШ у системі моніторингу мережевого трафіку дозволяє ефективно оптимізувати параметри моделі для виявлення вторгнень. Завдяки його здатності швидко адаптуватися до змінних умов, модель може динамічно уточнювати ваги ознак, пов’язаних із поведінковими характеристиками трафіку – такими як частота запитів, розмір пакетів, кількість з’єднань або часові патерни активності. Це підвищує чутливість системи до нетипових відхилень і забезпечує стабільну роботу навіть за наявності шуму в даних.

Отже, використання алгоритму БФГШ у задачах виявлення вторгнень створює передумови для розроблення адаптивних, високоточних моделей, здатних підтримувати баланс між швидкістю навчання, стабільністю та здатністю розпізнавати складні кіберзагрози в реальному часі.

Алгоритм градієнтного бустингу належить до ансамблевих методів машинного навчання, що будуються на принципі послідовного об’єднання слабких моделей у потужний прогнозувальний класифікатор. Його основна ідея полягає у тому, що кожна нова модель навчається на помилках попередніх, поступово

зменшуючи залишкову похибку та підвищуючи загальну точність системи. Такий підхід дозволяє ефективно моделювати складні нелінійні залежності у даних і досягати високих результатів навіть у випадках, коли інші методи машинного навчання демонструють обмежену ефективність.

Градієнтний бустинг ґрунтується на градієнтному спуску – ітераційному алгоритмі мінімізації функції втрат [40]. На кожному кроці створюється нове дерево рішень, що наближує негативний градієнт функції похибки, тобто навчається на залишках попереднього прогнозу. У результаті формується ансамбль дерев, зважених відповідно до їхнього внеску в зменшення помилки. Така архітектура забезпечує баланс між точністю та узагальнюючою здатністю моделі, що є особливо важливим для задач класифікації кіберзагроз, де дані часто містять шум і нерівномірно розподілені класи.

На рис. 2.2 наведено структурну схему роботи алгоритму градієнтного бустингу. Початково на вхід системи подається набір даних, який розподіляється на підмножини $D_1, D_2, ..., D_n$.

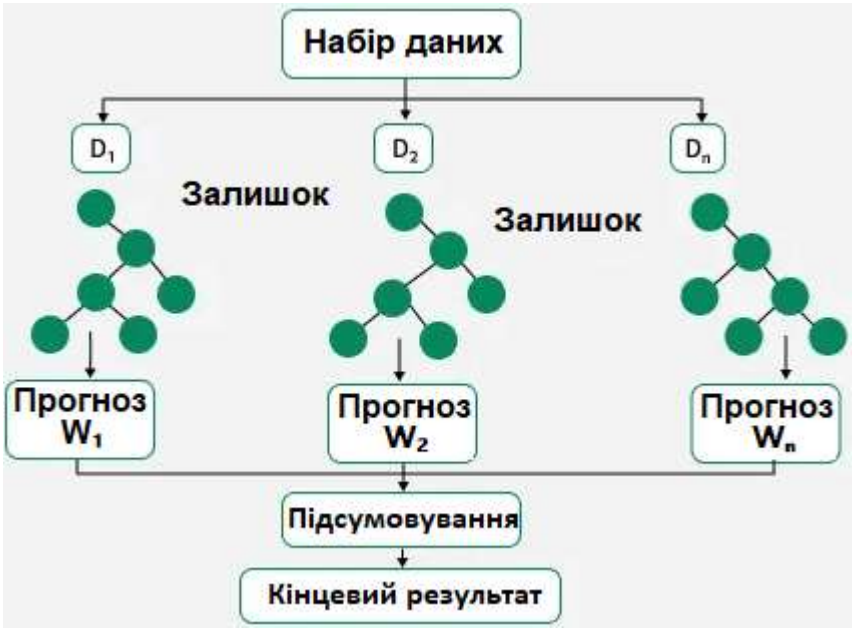

Рис. 2.2 Схема роботи алгоритму градієнтного бустингу [41]

Для кожної з них формується слабкий прогнозувальний алгоритм – дерево рішень, що генерує відповідний прогноз $W_1, W_2, ..., W_n$. Кожне наступне дерево навчається на залишковій похибці попередніх моделей, поступово уточнюючи прогноз. Після цього результати всіх моделей підсумовуються, формуючи кінцевий

результат – зважений прогноз системи. Такий підхід забезпечує поступове вдосконалення якості класифікації без перенавчання.

У системах моніторингу та виявлення вторгнень у мережевому трафіку градієнтний бустинг використовується для аналізу багатовимірних ознак – таких як частота пакетів, інтервали між запитами, розмір переданих даних та інші статистичні показники. Алгоритм здатний автоматично визначати найінформативніші параметри, створюючи гнучку модель, що виявляє аномальні шаблони поведінки, притаманні мережевим атакам. Його адаптивність дозволяє виявляти як відомі, так і нові типи загроз, у тому числі низькоінтенсивні або приховані атаки, які складно ідентифікувати традиційними методами.

Отже, впровадження градієнтного бустингу у процеси кіберзахисту відкриває можливість створення інтелектуальних систем аналізу трафіку, що поєднують точність класифікації з високою швидкодією прийняття рішень. Це формує основу для побудови адаптивних моделей безпеки, здатних оперативно реагувати на зміни в мережевому середовищі та підтримувати стабільність інформаційної інфраструктури навіть за умов зростання обсягів і складності атак.

Алгоритм дерева рішень є одним із базових, але водночас надзвичайно ефективних методів машинного навчання, який використовується як для задач класифікації, так і для регресії [42]. Його головна ідея полягає у послідовному поділі простору вхідних даних на підмножини за певними критеріями (ознаками), що дозволяє отримати модель у вигляді деревоподібної структури. Кожна внутрішня вершина дерева відповідає тесту на одну з ознак, кожна гілка – результату цього тесту, а листові вузли – кінцевим рішенням або класом прогнозу.

Такий підхід дозволяє відтворювати процес прийняття рішень людиною, тому дерево рішень часто розглядається як інтерпретована модель, що забезпечує прозорість і наочність процесу класифікації. На практиці алгоритми цього типу широко застосовуються в галузях кібербезпеки, медицини, фінансового моніторингу, енергетики та прогнозування ризиків. Вони здатні обробляти великі обсяги різнорідних даних і формувати чіткі правила прийняття рішень, що робить їх корисними для побудови систем підтримки аналітичних рішень і виявлення

аномалій.

На рис. 2.3 зображено узагальнену структуру роботи алгоритму дерева рішень. Початково система отримує набір даних, який використовується для побудови першого дерева, що розділяє простір спостережень за певною ознакою. Далі алгоритм послідовно уточнює моделі, навчаючи нові дерева на залишках прогнозу, тобто на помилках попередніх класифікацій. Кожне дерево в ансамблі – це самостійний класифікатор, який приймає власне рішення, а кінцевий прогноз формується на основі підсумовування або голосування всіх моделей. У результаті утворюється ансамбль дерев рішень, який забезпечує високу точність прогнозування та зменшує ймовірність перенавчання.

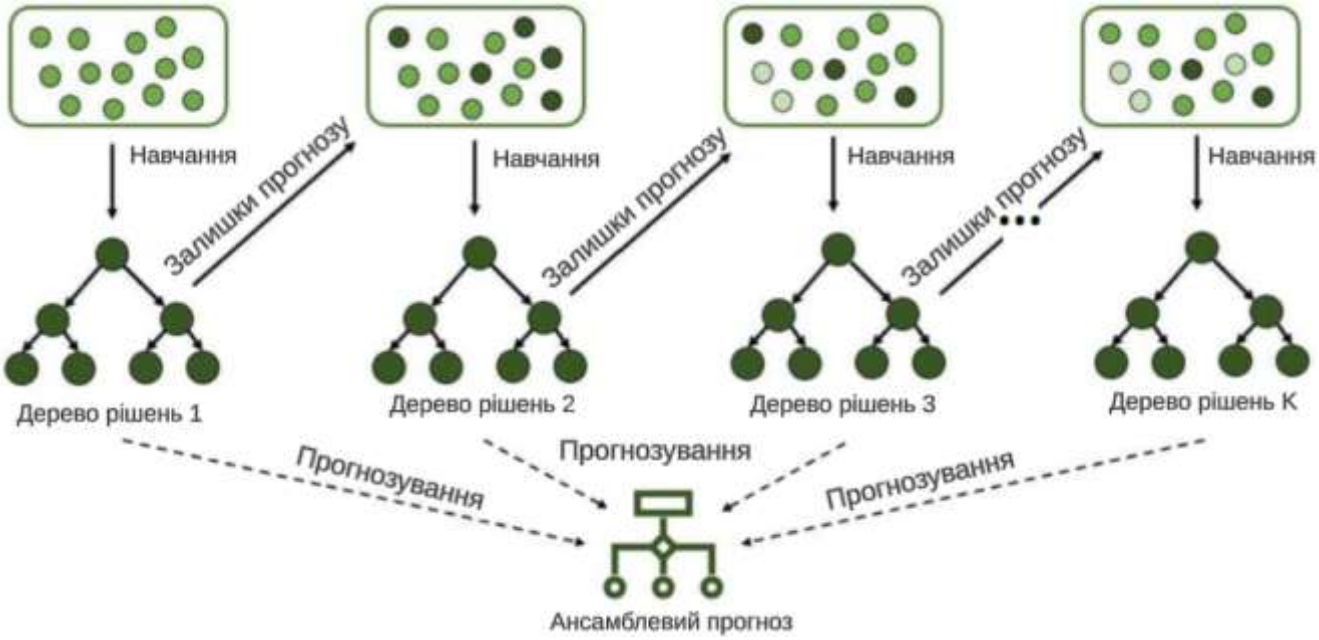

Рис. 2.3 Схема алгоритму дерева рішень [43]

Для задач моніторингу та виявлення вторгнень у мережевому трафіку дерева рішень використовуються для класифікації пакетів або сеансів зв’язку за їхніми характеристиками – такими як кількість з’єднань, обсяг переданих даних, часові інтервали, тип протоколу тощо. Кожен вузол дерева відображає певну умову (наприклад, «кількість з’єднань > 100»), а листові вершини вказують, чи належить подія до категорії нормальної активності або до потенційної атаки. Завдяки ієрархічній структурі дерево рішень може працювати в реальному часі, оперативно виявляючи відхилення у поведінці користувачів або систем. Крім того, цей алгоритм легко інтерпретується фахівцями з безпеки, що дає змогу не лише

фіксувати факт вторгнення, а й пояснювати причини його виявлення.

Використання алгоритмів дерева рішень у сфері кібербезпеки створює підґрунтя для побудови пояснюваних систем виявлення загроз, які поєднують високу точність із можливістю адаптації до нових умов мережевого середовища. Це забезпечує інтеграцію таких моделей у комплексні системи моніторингу, здатні не лише реагувати на атаки, але й передбачати їхні потенційні сценарії розвитку.

У табл. 2.1 наведено порівняльну характеристику зазначених алгоритмів за ключовими критеріями, що мають практичне значення для побудови інтелектуальної системи моніторингу мережевого трафіку.

Таблиця 2.1 – Порівняльна характеристика алгоритмів машинного навчання

Критерій оцінювання	БФГШ	Градієнтний бустинг	Дерево рішень
Тип алгоритму	Оптимізаційний, метод квазін'ютона	Ансамблевий, послідовне навчання слабких моделей	Ієрархічна класифікаційна модель
Швидкість збіжності	Висока завдяки апроксимації Гессе	Середня, залежить від кількості дерев	Висока для невеликих обсягів даних
Потреба в обчислювальних ресурсах	Помірна, ефективне використання пам'яті	Висока через послідовне навчання	Низька
Точність класифікації	Висока при оптимізації параметрів моделей	Дуже висока на великих наборах даних	Середня, залежить від глибини дерева
Інтерпретованість результатів	Середня, залежить від моделі, яку оптимізує	Низька, складно інтерпретувати ансамбль	Висока, легко пояснювана структура
Стійкість до шуму	Висока	Висока	Середня
Застосування у кібербезпеці	Оптимізація параметрів нейронних і ансамблевих IDS	Створення високоточних моделей виявлення аномалій	Побудова базових класифікаторів трафіку

Вибір методу Бройдена–Флетчера–Голдфарба–Шанно для розробки системи

обумовлений його здатністю забезпечувати швидку збіжність при оптимізації параметрів моделі навіть у багатовимірних просторах даних. Завдяки використанню апроксимації Гессе алгоритм дозволяє ефективно мінімізувати функцію втрат без значних обчислювальних витрат, що є критично важливим для задач моніторингу в реальному часі. Його стійкість до шуму та здатність адаптивно коригувати ваги моделей робить БФГШ надійним інструментом для виявлення складних аномалій у мережевому трафіку. Крім того, метод добре інтегрується з нейронними та ансамблевими підходами, забезпечуючи баланс між точністю класифікації та стабільністю роботи системи.

2.2 Методика підготовки даних та формування навчальних і тестових вибірок

Для реалізації інтелектуальної системи виявлення вторгнень використано відкритий набір даних NSL-KDD, розміщений на платформі Kaggle [44]. Цей набір є вдосконаленою версією класичного датасету KDD Cup 99 і призначений для навчання та тестування систем виявлення аномалій у мережевому трафіку. Його відмінною особливістю є усунення дубльованих записів і балансування класів, що підвищує репрезентативність вибірок і зменшує ризик перенавчання моделей.

Завантаження даних здійснено у форматі CSV із 41 вхідним атрибутом і цільовою змінною labels, що описує тип трафіку (нормальний або атакувальний). Кожен запис містить набір параметрів, які характеризують окреме мережеве з’єднання – від базових показників, таких як тривалість сеансу (duration) і тип протоколу (protocol_type), до статистичних індикаторів взаємодії (count, srv_count) та поведінкових характеристик користувача (logged_in, num_failed_logins).

Попередня обробка даних охоплювала кілька етапів. На першому етапі виконано очищення набору від пропущених або некоректних значень, а також приведення категоріальних ознак (наприклад, protocol_type, service, flag) до числового формату за допомогою методу Label Encoding. Другий етап передбачав нормалізацію числових параметрів, що дозволило уникнути домінування окремих ознак під час навчання моделі. Для забезпечення адекватного порівняння

результатів дані було розділено на навчальну (70 %) та тестову (30 %) вибірки з урахуванням пропорційності класів атак і нормального трафіку.

Використання датасету NSL-KDD є обґрунтованим, оскільки він містить різноманітні категорії кіберзагроз – DoS, Probe, R2L та U2R, що дозволяє моделі навчатися на комплексних прикладах поведінки зловмисників. Така структура даних дає змогу не лише класифікувати мережеві події, а й формувати адаптивні профілі трафіку для виявлення невідомих типів атак у майбутньому.

У табл. 2.2 наведено опис основних атрибутів тренувального набору даних, їх типів і функціонального призначення у моделі.

Таблиця 2.2

Атрибути набору даних CyberFedDefender

№	Назва атрибуту	Тип даних	Опис
1	2	3	4
1	duration	numeric	Тривалість мережевого з’єднання у секундах
2	protocol_type	categorical	Тип протоколу (tcp, udp, icmp)
3	service	categorical	Тип сервісу, через який здійснюється з’єднання
4	flag	categorical	Стан з’єднання після завершення (наприклад, SF, REJ)
5	src_bytes	numeric	Кількість байтів, відправлених від джерела до приймача
6	dst_bytes	numeric	Кількість байтів, отриманих від приймача
7	count	numeric	Кількість з’єднань із даним хостом за останні 2 секунди
8	srv_count	numeric	Кількість з’єднань із поточним сервісом за останні 2 секунди
9	serror_rate	float	Частка з’єднань із помилкою SYN
10	diff_srv_rate	float	Частка з’єднань із різними сервісами
11	dst_host_srv_count	numeric	Кількість з’єднань із певним сервісом на одному хості
12	dst_host_same_srv_rate	float	Частка з’єднань до одного сервісу на одному хості

13	num_failed_logins	numeric	Кількість невдалих спроб входу користувача

Продовження таблиці 2.1.

1	2	3	4
14	logged_in	binary	Індикатор успішної авторизації користувача
15	num_compromised	numeric	Кількість скомпрометованих умов у сеансі
16	root_shell	binary	Ознака отримання доступу до root
17	dst_host_serror_rate	float	Частка з'єднань із помилками SYN для цільового хосту
18	dst_host_srv_rerror_rate	float	Частка помилок з'єднання на рівні сервісу
19	labels	categorical	Клас події (normal або тип атаки: DoS, Probe, R2L, U2R)
20	hot	numeric	Кількість «гарячих» показників – підозрілих дій у сеансі
21	num_root	numeric	Кількість запитів на права адміністратора (root)
22	num_file_creations	numeric	Кількість створених файлів під час сеансу
23	num_shells	numeric	Кількість запущених оболонок командного рядка
24	num_access_files	numeric	Кількість звернень до критичних системних файлів
25	num_outbound_cmds	numeric	Кількість зовнішніх команд, відправлених за межі мережі (для FTP-з'єднань)
26	is_host_login	binary	Ознака, чи входив користувач як локальний (1 – так, 0 – ні)
27	is_guest_login	binary	Ознака, чи здійснено вхід як гість
2	srv_serror_rate	float	Частка з'єднань поточного сервісу з помилками

8			SYN
29	rerror_rate	float	Частка з’єднань із помилками відповіді (RST)
30	srv_rerror_rate	float	Частка помилок RST серед з’єднань поточного сервісу
31	same_srv_rate	float	Відношення кількості з’єднань до того самого сервісу
32	diff_srv_rate	float	Відношення кількості з’єднань до різних сервісів
33	srv_diff_host_rate	float	Частка з’єднань поточного сервісу з різними хостами

Продовження таблиці 2.1.

1	2	3	4
33	srv_diff_host_rate	float	Частка з’єднань поточного сервісу з різними хостами
34	dst_host_count	numeric	Кількість унікальних хостів, до яких здійснювалися з’єднання
35	dst_host_diff_srv_rate	float	Частка з’єднань до різних сервісів на одному хості
36	dst_host_same_src_port_rate	float	Частка з’єднань із того самого порту джерела до цільового хосту
37	dst_host_srv_diff_host_rate	float	Частка сервісних з’єднань до різних хостів
38	dst_host_srv_serror_rate	float	Частка помилок SYN для сервісу на цільовому хості
39	dst_host_rerror_rate	float	Частка з’єднань з помилками RST на цільовому хості
40	wrong_fragment	numeric	Кількість фрагментів пакету, що не відповідають стандарту TCP/IP
41	urgent	numeric	Кількість «термінових» пакетів у межах сеансу
42	su_attempted	binary	Ознака спроби підвищення прав користувача до root (1 – так, 0 – ні)
4	land	binary	Індикатор атак типу LAND, коли джерело та

3			приймач мають однакові IP-адреси
44	dst_host_serror_rate	float	Частка помилок SYN серед усіх з’єднань до певного хосту
45	dst_host_srv_count	numeric	Кількість з’єднань із поточним сервісом до цільового хосту
46	dst_host_srv_rerror_rate	float	Частка помилок RST серед сервісних з’єднань до певного хосту

Сформована методика підготовки даних забезпечує достовірність і збалансованість вибірок, що є критичним чинником для навчання алгоритмів машинного навчання в задачі виявлення вторгнень. Застосування структурованого процесу попередньої обробки дозволяє створити основу для подальшої побудови ефективної аналітичної моделі моніторингу мережевого трафіку.

Аналіз попередніх етапів підготовки даних дозволив не лише структурувати атрибути, а й дослідити взаємозв’язки між ними, що є критичним для формування збалансованої навчальної вибірки. Для цього побудовано матрицю кореляцій за коефіцієнтом Пірсона, яка відображає силу та напрям статистичних залежностей між основними показниками набору даних (рис. 2.4). Це дає змогу виявити надлишкові або слабоінформативні ознаки, оптимізувати обсяг вхідних параметрів та уникнути мультиколінеарності під час навчання моделі.

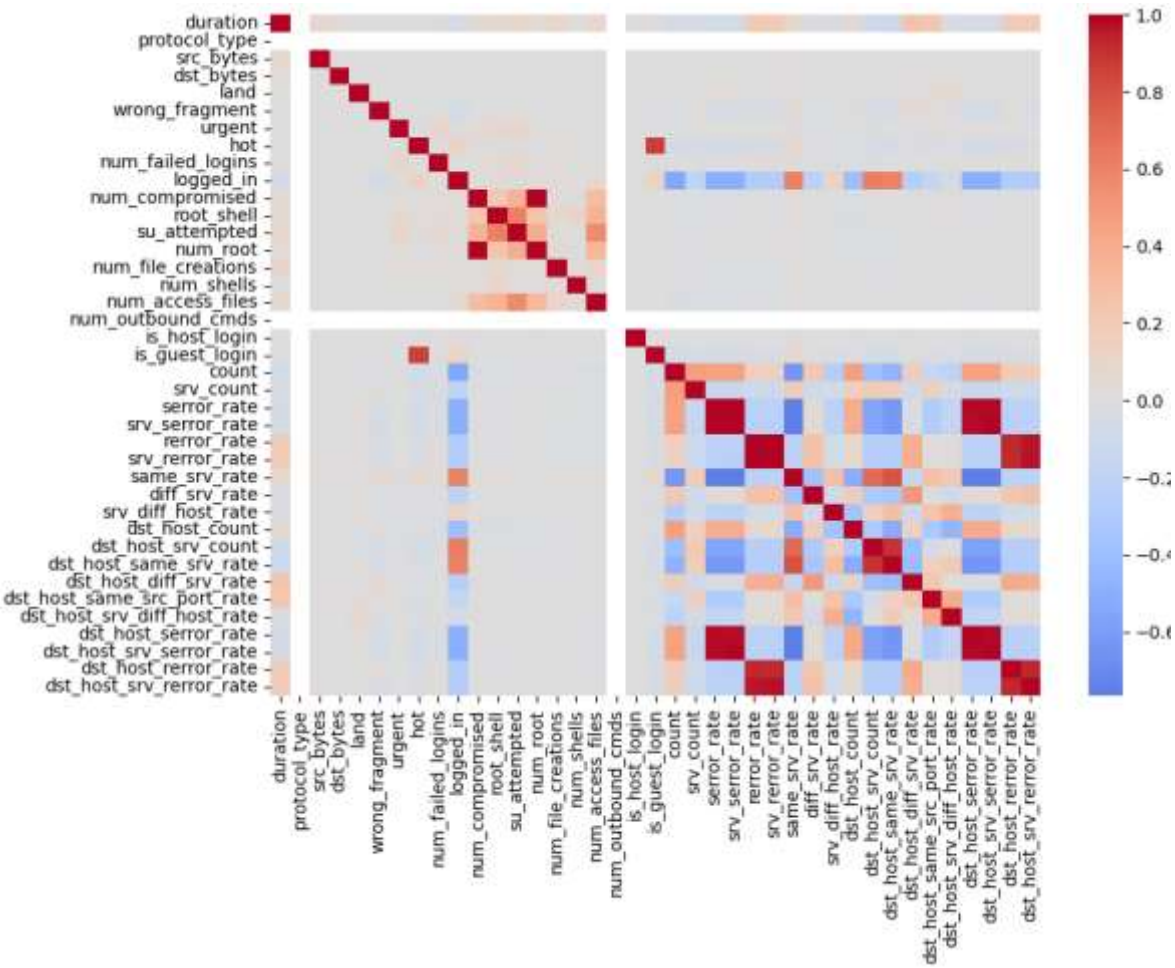

Рис. 2.4 Матриця кореляцій за Пірсоном

Найбільш сильна позитивна кореляція спостерігається між показниками srv_count, dst_host_srv_count і same_srv_rate, що свідчить про їхню взаємозалежність у контексті кількості однотипних з'єднань на рівні сервісів. Високі значення цих коефіцієнтів зазвичай вказують на нормальну роботу серверів, тоді як різке зниження або надмірне зростання кореляції може сигналізувати про аномальні стани мережі чи спроби атаки типу Denial of Service (DoS). Натомість показники rerror_rate та srv_rerror_rate демонструють середній рівень позитивного зв'язку, що відображає характер помилок у з'єднаннях, які часто супроводжують сканування портів або спроби несанкціонованого доступу.

У свою чергу, слабка або негативна кореляція спостерігається між параметрами, які характеризують різні рівні протоколів, наприклад між src_bytes і dst_bytes, що пояснюється асиметрією обміну даними у більшості мережевих транзакцій. Такий розподіл є типовим для реального трафіку, де запити та відповіді мають різний розмір пакетів. Деякі атрибути, як-от wrong_fragment, urgent,

num_outbound_cmds, практично не мають кореляції з іншими змінними, що свідчить про їхню унікальність та потенційну цінність для розпізнавання специфічних типів атак.

На рис. 2.5 зображено теплову карту кумулятивної густини, яка відображає спільну динаміку двох ключових параметрів – num_compromised (кількість скомпрометованих умов у сеансі) та count (кількість з'єднань із даним хостом за останні 2 секунди). Такий підхід дозволяє не лише оцінити ступінь концентрації значень у просторі атрибутів, але й виявити зони потенційних аномалій, що мають відхилення від типового розподілу.

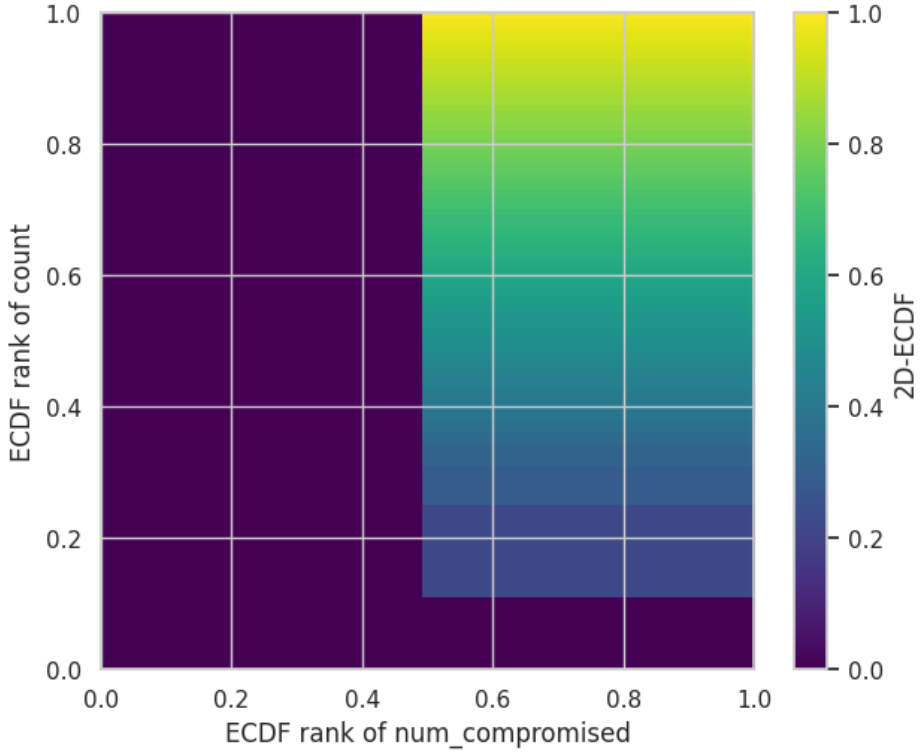

Рис. 2.5 Теплова карта кумулятивної густини

З аналізу рисунка видно, що висока щільність розподілу спостерігається у верхньому правому секторі теплової карти, де значення обох атрибутів мають високі ранги. Це свідчить про накопичення спостережень, у яких велика кількість однотипних з'єднань супроводжується значним числом скомпрометованих дій – характерна ознака атак типу Denial of Service (DoS) або Probe. Нижчі рівні щільності (відображені холодними тонами) відповідають нормальній поведінці мережі, де кількість з'єднань і скомпрометованих умов залишається у межах статистичної норми.

Рис. 2.6 відображає радар середніх нормалізованих ознак, який ілюструє характерні відмінності між нормальним мережевим трафіком та трафіком, що має ознаки атаки. Така форма подання даних дає змогу наочно оцінити відносну вагомість кожного параметра та його внесок у процес класифікації, що є важливим при побудові моделей машинного навчання для виявлення аномалій.

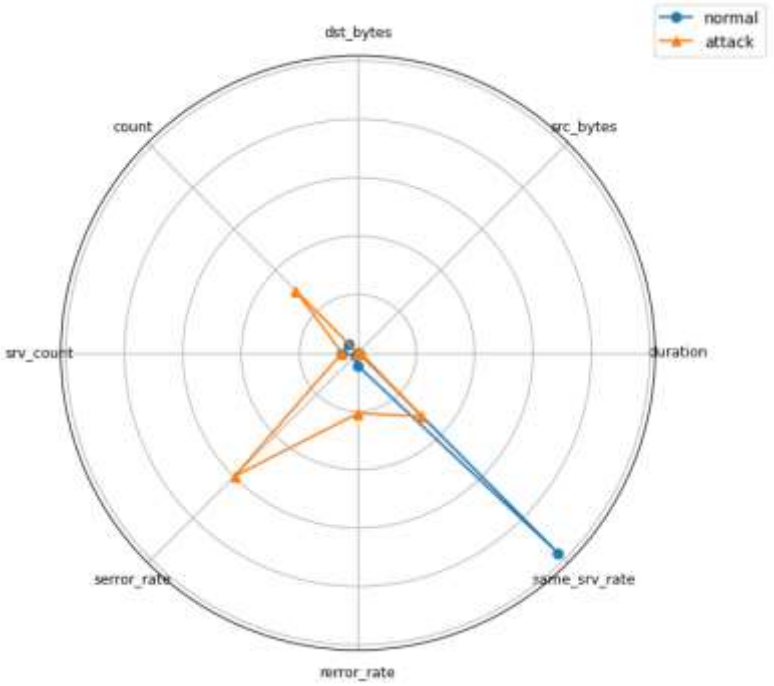

Рис. 2.6 Радар середніх нормалізованих ознак

На радарній діаграмі синя лінія відображає середні значення нормальних з’єднань, тоді як помаранчева – характеристику аномальної активності. Найбільш виразна різниця спостерігається за показниками same_srv_rate та serror_rate: для нормального трафіку перший параметр має значно більші значення, що свідчить про сталість обслуговування одним і тим самим сервісом, у той час як під час атак цей показник помітно знижується через множинні запити до різних портів чи хостів. Натомість serror_rate і count істотно зростають у випадку шкідливої активності, що відображає збільшення кількості помилкових або некоректно завершених з’єднань, характерних для DoS- або Probe-атак.

Значення src_bytes і dst_bytes демонструють відносну стабільність у нормальному режимі, тоді як під час атак часто спостерігається їхній дисбаланс – зростання обсягу вихідних даних при мінімальній кількості отриманих відповідей. Така поведінка є типовою для спроб несанкціонованого доступу або

перевантаження мережевих ресурсів.

На рис. 2.7 представлено Лоренц-криву для показника dst_bytes, яка відображає нерівномірність розподілу обсягів переданих даних між окремими зразками мережевого трафіку. Дана візуалізація є класичним інструментом статистичного аналізу, що дозволяє оцінити ступінь концентрації або дисбалансу у вибірці, тобто визначити, наскільки рівномірно розподіляється обсяг переданої інформації між сеансами.

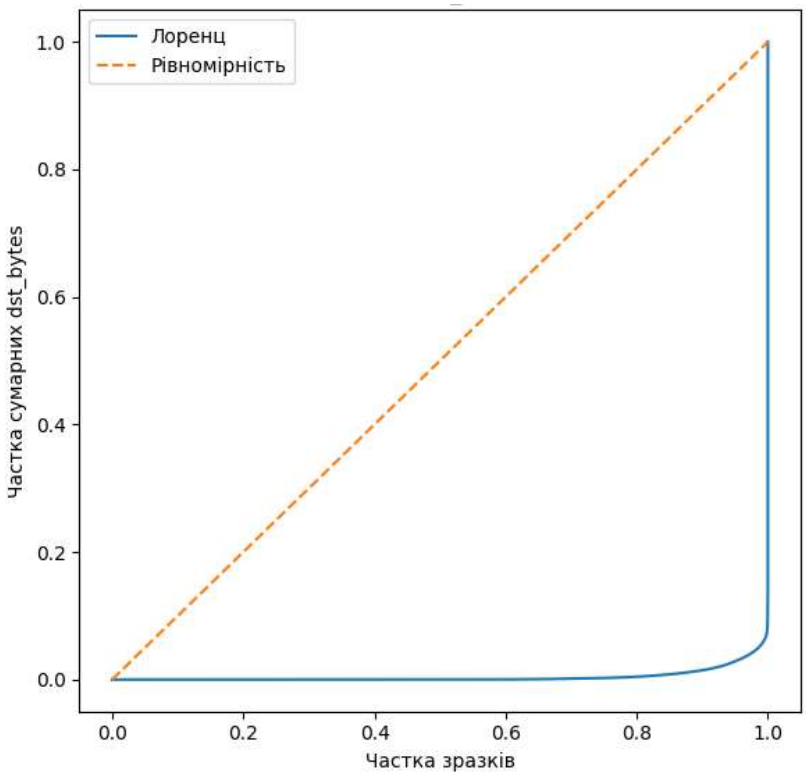

Рис. 2.7 Лоренц-крива для dst_bytes

Помаранчева пунктирна лінія демонструє ідеальний стан рівномірності, коли всі з'єднання передають однаковий обсяг даних, тоді як синя лінія Лоренца відображає фактичний розподіл. Значне відхилення кривої Лоренца вниз від лінії рівності свідчить про наявність високої асиметрії: більшість сеансів передають дуже незначну кількість байтів, тоді як невелика частка з'єднань формує основний трафік. Така поведінка характерна для реальних мереж, у яких наявні короткотривалі запити поряд із великими потоками даних, але може також вказувати на аномальну активність, пов'язану з атаками типу DoS або масовими передачами даних при компрометації системи.

Отримана форма кривої підтверджує доцільність подальшого аналізу

розподілу ознак у різних класах трафіку, зокрема для ідентифікації потенційних «пікових» джерел активності. У межах системи моніторингу така аналітика дозволяє налаштувати адаптивні порогові значення, що забезпечують більш точне відокремлення аномальних патернів від нормальної поведінки, підвищуючи чутливість і надійність процесу виявлення вторгнень.

На рис. 2.8 подано залежність частки атак від показника dst_bytes, отриману в результаті квантильного бінування. Такий підхід дає змогу оцінити, як змінюється ймовірність появи атак у різних діапазонах обсягів переданих даних до вузла призначення. Поділ значень на квантилі забезпечує коректне порівняння між зонами із різною щільністю спостережень, дозволяючи виявити нелінійні закономірності, притаманні аномальній поведінці трафіку.

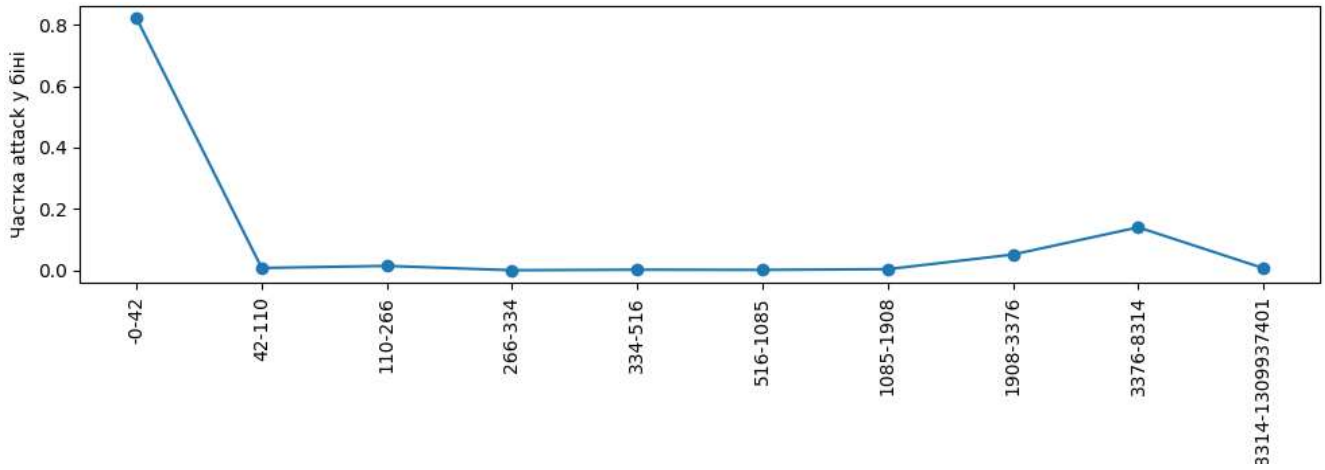

Рис. 2.8 Залежність частки атак від dst_byte (квантильне бінування)

З аналізу рисунка видно, що переважна більшість атак зосереджена в інтервалі малих значень dst_bytes – до приблизно 40–50 байтів. У цій області частка шкідливих з’єднань перевищує 80 %, що свідчить про характерну ознаку атак типу DoS або Probe, для яких типовими є короткі запити з мінімальним обсягом відповідей від цільового хоста. У подальших бінових інтервалах спостерігається різке зниження частки атак і стабілізація показників поблизу нуля, що відповідає звичайним користувацьким з’єднанням із типовими обсягами переданих даних. Лише в зоні середніх і вищих квантилів відмічено незначне підвищення частки атак, яке може бути пов’язане зі складнішими формами вторгнень, зокрема зі спробами передачі виконуваного коду або експлуатації сервісів з високою пропускною здатністю.

Рис. 2.9 висвітлює графічне відображення найбільш корельованих ознак із бінарною міткою label_binary, яка позначає належність зразків до класів «нормальний трафік» або «атака». Аналіз здійснено за модулем коефіцієнта кореляції Пірсона, що дозволяє визначити ступінь лінійної залежності між окремими характеристиками мережевих з'єднань і фактом наявності вторгнення.

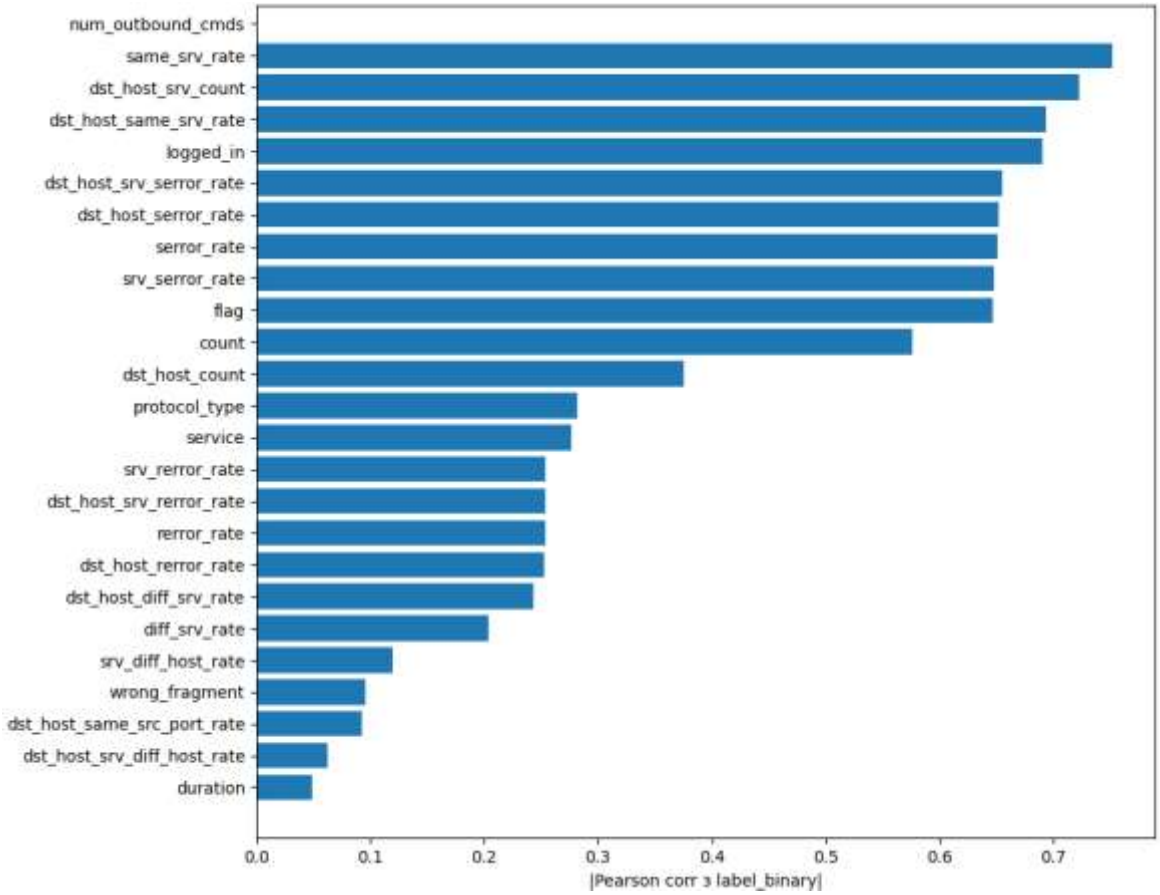

Рис. 2.9 Найбільш корельовані ознаки

З отриманих результатів видно, що найбільш значущими змінними є num_outbound_cmds, same_srv_rate, dst_host_srv_count і dst_host_same_srv_rate, які демонструють найвищі значення кореляції (0.65–0.75). Це вказує на те, що кількість вихідних команд, стабільність обслуговування одним сервісом і частота звернень до певного хосту мають суттєвий вплив на визначення шкідливої активності. Високі значення цих параметрів зазвичай є характерними для легітимних сеансів, тоді як їхні різкі зміни чи зменшення часто супроводжують атаки типу Probe або DoS.

Дещо нижчий, але також помітний вплив мають показники serror_rate, srv_serror_rate, dst_host_serror_rate та flag, що відображають рівень помилок на

рівні сервісів і протоколів. Підвищення цих значень зазвичай свідчить про нестандартну поведінку або порушення звичайного циклу обміну пакетами, що може бути ознакою агресивного сканування або спроби перевантаження. Натомість атрибути на кшталт wrong_fragment, diff_srv_rate чи duration мають відносно низьку кореляцію, що робить їх другорядними для моделі.

На рис. 2.10 наведено співставлення середніх значень dst_bytes для найпоширеніших сервісів із часткою атак, що відбувалися під час роботи цих протоколів. Такий підхід дозволяє поєднати кількісний аспект обміну даними з якісною характеристикою безпеки кожного сервісу, що є важливим для розуміння поведінкових закономірностей у мережевому трафіку.

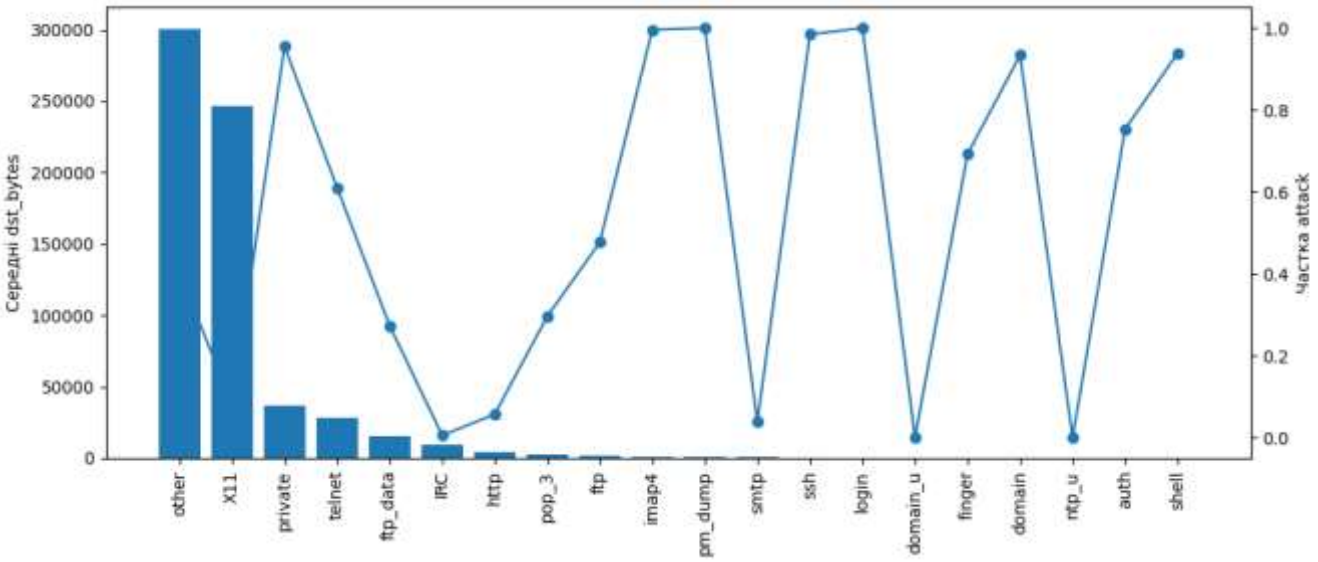

Рис. 2.10 ТОП-20 сервісів: середні dst_bytes та частка атак

Згідно з отриманими результатами, найвищі середні значення dst_bytes спостерігаються для сервісів other та X11, які характеризуються значними обсягами переданих даних. Це пояснюється тим, що зазначені протоколи часто використовуються у високорівневих клієнт-серверних взаємодіях або як канали для складних додатків. Водночас для сервісів ftp_data, IRC, http і ftp середні показники переданих байтів є відносно низькими, але саме в цих сегментах помітно підвищується частка атак, що свідчить про активне використання таких протоколів зловмисниками для експлуатації вразливостей або передачі шкідливого контенту.

Особливої уваги заслуговують сервіси smtp, domain, auth і shell, для яких зафіксовано майже стовідсоткову частку атак. Такі результати узгоджуються з

практичними спостереженнями у сфері кібербезпеки: подібні сервіси часто є об’єктами атак через автентифікаційні механізми, відкриті порти та спрощені схеми обміну повідомленнями. З іншого боку, протоколи ssh та login демонструють мінімальний рівень атак при відносно стабільних середніх обсягах даних, що вказує на їхню більшу захищеність та контрольований доступ.

Рис. 2.11 висвітлює тривимірну поверхню частки атак, побудовану у просторі ознак same_srv_rate та dst_host_srv_count. Зокрема, змінна same_srv_rate характеризує частку запитів, що спрямовані до одного й того ж сервісу, тоді як dst_host_srv_count відображає кількість сервісів, з якими взаємодіє цільовий хост.

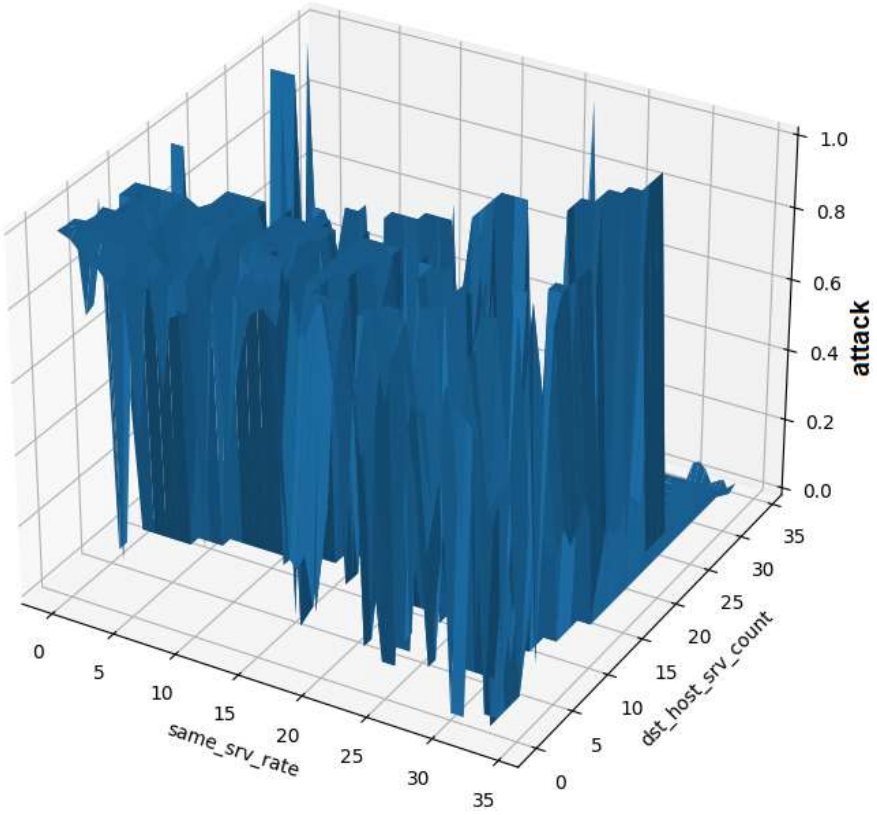

Рис. 2.11 3D-поверхня частки атаки

Графік показує складну нерівномірну поверхню з численними піками та спадами, що свідчить про наявність нелінійних залежностей між цими показниками та наявністю вторгнень. Найбільша щільність атак спостерігається у зонах низьких значень same_srv_rate при середніх і високих значеннях dst_host_srv_count. Це вказує на те, що атаки частіше трапляються в умовах, коли потік запитів розподіляється між великою кількістю сервісів, але без сталого характеру взаємодії – типовий патерн для сканувальних і розподілених DoS-атак. Навпаки, області з

високими same_srv_rate демонструють зменшення ймовірності атак, що відповідає стабільним користувацьким сесіям, де комунікація зосереджена на одному або кількох перевірених сервісах.

Отримана просторова модель дозволяє зробити висновок про доцільність використання цих двох ознак у складі класифікаційної системи, орієнтованої на виявлення аномалій. Поєднання same_srv_rate і dst_host_srv_count відображає баланс між інтенсивністю та розмаїттям запитів, що є ключовим фактором для диференціації між нормальними та підозрілими мережевими з'єднаннями. Така форма аналізу підсилює інтерпретованість моделей машинного навчання, забезпечуючи візуальне розуміння динаміки виникнення атак у багатовимірному просторі ознак.

2.3 Математична модель та архітектура інтелектуальної системи моніторингу мережевого трафіку

Розроблення моделі базується на положеннях теорії машинного навчання, де процес виявлення загроз формулюється як задача багатокласової або бінарної класифікації. У цьому контексті основною метою є побудова функції відображення, що на основі сукупності параметрів з'єднання дозволяє з високою точністю визначити, чи належить потік до нормального трафіку, чи містить ознаки вторгнення. Для цього використовуються різні типи ознак – як безпосередньо вимірювані кількісні параметри, так і похідні індикатори, отримані в результаті аналітичної обробки даних.

Важливою складовою архітектури системи є модуль попередньої обробки, який здійснює нормалізацію, перетворення категоріальних змінних та усунення шумів у даних. Далі інформація надходить до навчальної моделі, де виконується побудова класифікаційної функції, що узагальнює закономірності поведінки мережі. Такий підхід дозволяє не лише автоматизувати процес моніторингу, а й створити основу для самонавчання системи, яка з часом підвищує точність розпізнавання атак.

На початковому етапі система отримує дані про мережеві з'єднання, які

характеризуються як числовими параметрами (тривалість, обсяг переданих байтів, кількість пакетів тощо), так і категоріальними атрибутами (тип протоколу, статус з'єднання, службові порти). Для формалізації цих даних вводимо вектор:

$$x=\left(x^{(num)}, x^{(cat)}\right) \in R^p \times C^q, \quad (2.1)$$

де $x^{(num)}$ – підмножина числових ознак, а $x^{(cat)}$ – категоріальні характеристики, які належать до множини C. Така репрезентація дозволяє об'єднати різнорідні параметри трафіку в єдиний простір для подальшого машинного аналізу. Оскільки алгоритми машинного навчання працюють у числовому просторі, категоріальні змінні потребують спеціального відображення. Для цього використовується one-hot кодування, яке переводить кожну категорію у бінарний вектор фіксованої довжини:

$$\psi(c) \in \{0,1\}^{m(c)}, \quad (2.2)$$

де $m(c)$ – кількість можливих значень для конкретної змінної c, а одиниця вказує на активну категорію. Це перетворення усуває ординальність і забезпечує рівновіддаленість усіх категорій у метричному просторі.

Після кодування кожної категоріальної змінної формується повний вектор ознак, що відображає комбінацію всіх числових і закодованих параметрів. Це відображення визначається функцією:

$$\phi(x)=\left[x^{(num)}; \psi(c_1); \ldots; \psi(c_q)\right] \in R^d, \quad (2.3)$$

де: d – сумарна кількість ознак після кодування. Таким чином, функція $\phi(x)$ виконує роль оператора ознакового простору, що приводить усі типи даних до уніфікованого числового вигляду, придатного для обчислювальних моделей.

Для запобігання впливу масштабних диспропорцій між різними параметрами трафіку застосовується мінімаксна нормалізація. Вона перетворює кожну числову ознаку у відрізок [0,1] за формулою:

$$z=\frac{z-min(z)}{max(z)-min(z)}, z \in [0,1]. \quad (2.4)$$

Нормалізація гарантує стабільність процесу навчання моделі та прискорює збіжність оптимізаційних алгоритмів. У контексті мережевого моніторингу це означає, що незалежно від діапазону трафіку (наприклад, для великих і малих потоків даних) усі характеристики набувають рівновагомого внеску під час аналізу.

Для побудови моделі необхідно формально визначити структуру навчальних даних. Нехай задано множину спостережень:

$$D_{tr}=\{(x_i, y_i)\}_{i=1}^{n_{tr}} \quad (2.5)$$

де x_i – вектор ознак, а $y_i \in Y$ – відповідна мітка класу, що описує стан мережевої активності (нормальний або один із типів атак). Аналогічно задається тестова множина D_{tr} для оцінювання узагальнюючої здатності моделі. Це створює основу для подальшої побудови оптимізаційної задачі, у якій модель навчається розрізняти нормальні та аномальні стани трафіку.

Після формування нормалізованого вектора ознак $\phi(x)$, кожному класу $k \in \{1, \dots, K\}$ ставиться у відповідність лінійна дискримінантна функція:

$$f_k(x)=(w_k, \phi(x))+b_k, \quad (2.6)$$

де $w_k \in R^d$ – вектор вагових коефіцієнтів, що характеризує вплив окремих ознак на приналежність до класу k, а b_k – зміщення (поріг активації).

Оскільки система має справу з багатокласовим середовищем (різні типи атак), кожна підзадача класифікації перетворюється на двійкову постановку «один проти решти». Для кожного класу k визначимо допоміжну змінну:

$$y_{ik}=\begin{cases} +1, \text{якщо } y_i=k \\ -1, \text{якщо } y_i \neq k \end{cases}. \quad (2.7)$$

Таке представлення дозволяє побудувати окремий лінійний роздільник для кожного типу мережевої активності. Надалі система об’єднує результати всіх підзадач, що забезпечує здатність класифікувати події у багатовимірному середовищі трафіку.

Щоб знайти оптимальні параметри моделі (w_k, b_k), розв’язується задача мінімізації регуляризованої функції втрат. Для методу стохастичного двоїстого координатного спуску (СДКС) вона може бути записана у вигляді:

$$L(W, b)=\frac{1}{n_{tr}}\sum_{i=1}^{n_{tr}}\sum_{k=1}^{K} max\left\{0,1-y_{ik}\left((w_k, \phi(x))+b_k\right)\right\}+\frac{\lambda}{2}\sum_{k=1}^{K}\|w_k\|_2^2. \quad (2.8)$$

Перший доданок відображає гінґ-втрату, яка карає неправильну або недостатньо впевнену класифікацію. Другий доданок – регуляризаційний термін, що обмежує зростання вагових коефіцієнтів і тим самим запобігає перенавчанню моделі. Параметр $\lambda>0$ визначає компроміс між точністю навчання та

узагальнюючою здатністю.

У методі СДКС оптимізація проводиться не безпосередньо за ваговими коефіцієнтами, а за допомогою двоїстих змінних a_{ik}, які пов'язані з кожним спостереженням і класом. Зв'язок між двоїстими змінними та вагами задається виразом:

$$w_k = \sum_{i=1}^{n_{tr}} a_{ik} \cdot y_{ik} \cdot \phi(x_i). \quad (2.9)$$

Це означає, що кожен вектор ваг формується як зважена комбінація векторів ознак навчальних зразків. Фактично модель «запам'ятовує» репрезентативні приклади мережевих станів і використовує їх для розмежування класів. Такий підхід забезпечує високу обчислювальну стабільність навіть на великих обсягах трафіку.У реальних мережевих даних частка нормального трафіку, як правило, значно перевищує кількість прикладів атак. Для компенсації цього ефекту вводяться ваги класів ωk, що дозволяють посилити внесок рідкісних, але критично важливих подій (наприклад, DDoS або фішинг-атак) у процес навчання. Модифікована функція втрат набуває вигляду:

$$L_w(W, b) = \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \sum_{k=1}^{K} w_k max\left\{0, 1 - y_{ik}\left((w_k, \phi(x)) + b_k\right)\right\} + \frac{\lambda}{2} \sum_{k=1}^{K} \|w_k\|_2^2. \quad (2.10)$$

Така адаптація дозволяє системі не втрачати чутливість до рідкісних атак, водночас підтримуючи збалансовану точність класифікації у всіх класах.

Після завершення навчання система отримує набір вагових коефіцієнтів $W=[w_1, w_2, \ldots, w_K]$ і зсувів b_k. Для нового зразка мережевого трафіку x розраховуються всі значення дискримінантних функцій $f_k(x)$. Клас, до якого належить потік, визначається правилом максимальної оцінки:

$$y(x) = argm_{k \in \{1,\ldots,K\}} ax f_k(x). \quad (2.11)$$

Це означає, що система відносить спостереження до того класу, для якого відгук моделі має найбільшу величину. У контексті моніторингу трафіку таке правило дозволяє оперативно ідентифікувати тип активності – наприклад, нормальну поведінку користувача чи характерну ознаку атаки.

Щоб забезпечити більш гнучке прийняття рішень, результати класифікації переводяться у ймовірнісний простір за допомогою нормування значень дискримінантних функцій через *Softmax*-перетворення:

$$p_k(x)=\frac{exp(\gamma f_k(x))}{\sum_{l=1}^{K} exp(\gamma f_l(x))}, \sum_{k=1}^{K} p_k(x)=1. \quad (2.12)$$

Параметр $\gamma>0$ виконує роль коефіцієнта масштабування (або «температури»), який контролює рівень «впевненості» моделі. Отримані значення $p_k(x)$ відображають імовірність того, що потік належить до певного класу. У системі моніторингу це дозволяє не лише визначити категорію, але й оцінити ступінь довіри до рішення – наприклад, сигналізувати про атаку лише тоді, коли $p_k(x)$ перевищує певний поріг.

Для кількісної оцінки ефективності навченої моделі використовуються показники точності.

Загальна (мікро-)точність розраховується як відношення кількості правильних прогнозів до загальної кількості спостережень:

$$MicroAcc=\frac{\{(x_i,y_i)\in D_{te}:y(x_j)=y_j\}}{n_{te}}. \quad (2.13)$$

Аналогічно, середня (макро-)точність визначається як середнє значення точності по кожному класу:

$$MicroAcc=\frac{1}{K}\sum_{k=1}^{K}\frac{TP_k}{TP_k+FN_k}, \quad (2.14)$$

де: TP_k (True Positives) – кількість правильно виявлених атак певного типу, а FN_k – кількість випадків, коли атака цього типу не була розпізнана. Ці показники дозволяють оцінити збалансованість класифікації та визначити, наскільки система однаково добре розпізнає як поширені, так і рідкісні загрози.

Для вимірювання загальної невизначеності прогнозів використовується логарифмічна функція втрат, яка оцінює відхилення прогнозованого розподілу ймовірностей від фактичних міток:

$$LogLoss=\frac{1}{n_{te}}\sum_{j=1}^{n_{te}} logp_{y_j}(x_j). \quad (2.15)$$

Цей критерій особливо важливий у задачах кібербезпеки, де навіть незначна похибка у визначенні ймовірностей може призвести до помилкових сигналів. Менше значення *LogLoss* свідчить про більш точне калібрування моделі, тобто про здатність системи коректно оцінювати рівень ризику кожної події.

Для порівняння ефективності моделі із базовим класифікатором, який завжди прогнозує найпоширеніший клас (нормальний трафік), вводиться коефіцієнт

зниження логарифмічної втрати:

$$LLR=1-\frac{LogLoss_{model}}{LogLoss_{null}}. \quad (2.16)$$

Якщо $LLR>0$, модель перевершує базовий підхід, а при $LLR\to 1$ її прогнози стають максимально інформативними. Таким чином, цей показник дає змогу формально оцінити, наскільки інтелектуальна система справді підвищує якість моніторингу порівняно з тривіальними методами.

Після навчання моделі потік мережевих подій описується часовою послідовністю $\{x_t\}_{t\geq 1}$. Для кожного спостереження обчислюється прогноз

$\hat{y}_t=\hat{y}_t(x_t)$. Інцидентом вважається будь-яке відхилення від нормального класу k_0:

$$I_t=1\{\hat{y}_t \neq k_0\}, \quad (2.17)$$

де $1\{\cdot\}$ – індикаторна функція, що набуває значення 1 при виявленні аномалії.

Таким чином, система автоматично маркує кожен момент часу як нормальний або підозрілий, забезпечуючи базовий рівень моніторингу мережевого середовища.

Окрім простої бінарної оцінки, доцільно враховувати ступінь впевненості моделі у зробленому прогнозі. Для цього вводиться метрика маржинального розриву між найвищим та другим за величиною відгуками дискримінантних функцій:

$$\Delta_t=f_{\hat{y}_t}(x_t)-\max_{k\neq y_t} f_k(x_t). \quad (2.17)$$

Якщо Δ_t є великою, то класифікація є достовірною; якщо ж вона наближається до нуля – рішення невпевнене. Такий підхід дозволяє фільтрувати випадкові коливання трафіку, не спричиняючи надлишкових хибних тривог.

Для відображення загальної тенденції появи інцидентів у часі застосовується експоненційне згладжування, яке дозволяє враховувати не лише поточний стан, а й історію попередніх подій:

$$\rho_t=(1-\beta)_{\rho_{t-1}}+\beta I_t,\ \rho_0 \in [0,1],\ \beta \in (0,1]. \quad (2.18)$$

Параметр β визначає вагу останніх спостережень – при більших значеннях система стає більш чутливою до змін. Отриманий показник ρ_t можна трактувати як індикатор поточної активності загроз, що відображає середню частоту інцидентів за останній період.

Щоб забезпечити автоматичне реагування на надмірну кількість підозрілих

подій, вводиться пороговий критерій сповіщення:

$$Alert_t=1\{\rho_t>\tau\}, \tag{2.19}$$

де $\tau \in (0,1)$ – заданий поріг інтенсивності атак. Якщо рівень ρ_t перевищує порогове значення, система ініціює оповіщення адміністратора безпеки або вживає автоматичних контрзаходів. Така схема дає змогу швидко локалізувати сплески підозрілої активності, не потребуючи ручного втручання.

Для підвищення точності фіксації вторгнень застосовується комбінований критерій, який враховує не лише сам факт відхилення від нормального класу, а й рівень впевненості моделі:

$$I_t^{(\gamma)}=(\mu_t\text{-}\mu_t\text{-}w)^T \cdot \left(\frac{1}{2}(\sum_t + \sum_{t\text{-}w})\right)^{-1} \cdot (\mu_t\text{-}\mu_t\text{-}w), \tag{2.20}$$

де μ_t та Σ_t – оцінки середнього вектора та коваріаційної матриці ознак у поточному вікні. Якщо значення T_t^2 перевищує порогове значення, система фіксує структурну зміну в трафіку, що може свідчити про новий тип загроз або аномалію у мережевій поведінці.

Для пріоритезації подій за рівнем ризику запроваджується маржинальний показник:

$$M_t=\max_k f_k(x_t)\text{-}median_k f_k(x_k), \tag{2.21}$$

який дозволяє оцінювати відхилення найсильнішого сигналу від середнього рівня. Чим більшим є M_t, тим більш вираженою є ознака вторгнення, і така подія отримує вищий пріоритет для подальшого аналізу чи автоматичної реакції.

Для формалізації безперервного процесу моніторингу вводиться марковська модель станів системи:

$$S_{t+1}=F(S_t, x_{t+1}, \hat{y}_t(x_{t+1}); \Theta), \tag{2.22}$$

де S_t – поточний стан системи, який включає накопичену статистику, індикатори інцидентів та активні сигнали; F – функція переходу, що описує зміну стану під впливом нового спостереження x_{t+1} та прогнозу $\hat{y}_t(x_{t+1})$; $\Theta\{W, b\}$ – параметри моделі. Цей вираз відображає здатність системи адаптуватися до змін середовища в режимі реального часу.

Заключним етапом є встановлення відповідності між виявленим класом події

та реакцією системи безпеки:

$$A_t = r(\hat{y}_t), \text{де } r(k_0) = \emptyset, r(k) = \emptyset \text{для } k \neq k_0. \quad (2.23)$$

Відображення $r: Y \rightarrow A$ задає набір дій (наприклад, ізоляція вузла, блокування IP-адреси, запис події у журнал), що автоматично активуються при виявленні певного типу вторгнення. Це завершує математичну схему повного циклу реагування інтелектуальної системи.

Розроблена математична модель інтелектуальної системи моніторингу мережевого трафіку охоплює весь аналітичний цикл – від перетворення та нормалізації даних до навчання багатокласової моделі, прогнозування й оцінки точності, а також моніторингу, виявлення та реакції на інциденти.

2.4 Обґрунтування вибору технологій для розробки системи

Розроблення інтелектуальної системи моніторингу мережевого трафіку потребує вибору технологічного стеку, який забезпечить оптимальне поєднання продуктивності, гнучкості, масштабованості та зручності інтеграції алгоритмів машинного навчання. Вибір інструментів програмування має базуватись не лише на їх технічних характеристиках, але й на можливості реалізації багаторівневої архітектури, сумісності з бібліотеками для аналітики та стабільності при роботі з великими обсягами даних. У контексті побудови програмних систем кібербезпеки найбільш доцільним є використання сучасних мов високого рівня, таких як Python, Java та C#, кожна з яких має свої сильні сторони та сфери ефективного застосування.

Python є динамічною мовою програмування, що вирізняється простотою синтаксису та широкою екосистемою бібліотек для машинного навчання, аналізу даних і візуалізації (зокрема, TensorFlow, Scikit-learn, Pandas, Matplotlib). Завдяки своїй гнучкості Python активно використовується для побудови прототипів, наукових обчислень і моделювання систем виявлення вторгнень [45]. Його інтеграція з середовищами обробки великих даних робить цю мову оптимальною для реалізації аналітичних модулів системи.

Java характеризується стабільністю, кросплатформеністю та високою

продуктивністю, що робить її придатною для створення масштабованих серверних застосунків у сфері мережевої безпеки [46]. Розвинена екосистема Java (Spring Framework, Weka, Deeplearning4j) дозволяє ефективно реалізовувати як класичні алгоритми оброблення даних, так і компоненти для розподіленої аналітики. Завдяки підтримці багатопоточності Java забезпечує надійну роботу системи при великій кількості одночасних запитів.

C# є універсальною мовою програмування, орієнтованою на розроблення інтерактивних програмних продуктів і систем управління інформаційними потоками [47]. Вона забезпечує тісну інтеграцію з технологіями Microsoft, зокрема ML.NET, що дозволяє створювати локальні та корпоративні рішення з вбудованими моделями машинного навчання. Використання C# у поєднанні з Windows Forms або WPF надає можливість створення інтуїтивно зрозумілого графічного інтерфейсу для моніторингу мережевого трафіку в реальному часі.

Для обґрунтованого вибору технологічного середовища розробки інтелектуальної системи доцільно здійснити порівняльний аналіз основних мов програмування, що застосовуються у сфері машинного навчання та кібербезпеки. Такий аналіз дозволяє визначити оптимальне поєднання параметрів, серед яких – продуктивність виконання, зручність розробки, рівень підтримки бібліотек для штучного інтелекту, можливості інтеграції з базами даних і наявність засобів створення графічного інтерфейсу. Порівняння мов програмування напведено у табл. 2.3, яка узагальнює як якісні, так і кількісні показники, що мають вирішальне значення при побудові систем моніторингу мережевого трафіку.

Таблиця 2.3

Порівняння мов програмування за основними характеристиками

№	Характеристика	Python	Java	C#
1	Продуктивність виконання (операцій/с)	~1,5 млн	~3,2 млн	~3,8 млн
2	Підтримка багатопоточності	Обмежена (через GIL)	Висока	Дуже висока (асинхронні Task API)
3	Зручність для машинного навчання	Дуже висока (Scikit-learn,	Середня (Weka, DL4J)	Висока (ML.NET, Accord.NET)

		TensorFlow)		
4	Середній обсяг пам’яті на процес (МБ)	45–60	35–50	25–40
5	Сумісність з Windows-екосистемою	Обмежена	Помірна	Повна (Microsoft .NET)
6	Рівень безпеки типізації та виконання	Середній	Високий	Дуже високий (CLR контроль пам’яті)
7	Зручність налагодження (debugging)	Висока	Висока	Дуже висока (Visual Studio Tools)
8	Середня кількість рядків коду для реалізації моделі ML	~80–100	~120–140	~60–70
9	Портативність	Висока	Дуже висока	Висока (через .NET Core)
10	Середня кількість активних бібліотек для ML/AI	>250	~120	~90
11	Простота інтеграції з GUI та аналітичними модулями	Середня	Середня	Висока

Виходячи з проведеного порівняльного аналізу, мову програмування C# обрано для розроблення інтелектуальної системи моніторингу мережевого трафіку як найбільш збалансоване рішення за критеріями продуктивності, стабільності та інтеграційних можливостей. Завдяки платформі .NET ця мова забезпечує високий рівень сумісності з операційними системами Windows, що є важливим для реалізації корпоративних рішень у сфері кібербезпеки. Наявність інструментарію ML.NET дозволяє безпосередньо вбудовувати алгоритми машинного навчання у програмний код без потреби в зовнішніх бібліотеках, що підвищує узгодженість системи та знижує її обчислювальні витрати. Окрім того, C# відзначається розвиненими засобами налагодження, підтримкою об’єктно-орієнтованого програмування та високою безпекою виконання, що гарантує стабільну роботу в умовах інтенсивного потокового аналізу даних. Сукупність цих властивостей робить C# оптимальним вибором для побудови ефективного аналітичного середовища моніторингу та виявлення мережевих загроз.

Для реалізації інтелектуальної системи моніторингу мережевого трафіку, створеної мовою C#, важливим етапом є вибір оптимального середовища розробки, яке забезпечить повноцінну підтримку екосистеми .NET, зручні інструменти налагодження, інтеграцію з бібліотеками машинного навчання та зручний графічний інтерфейс. Серед найбільш ефективних середовищ для роботи з технологіями Microsoft варто розглянути Visual Studio 2022, Visual Studio Code та JetBrains Rider, кожне з яких має власні переваги залежно від цілей і масштабів проєкту.

Visual Studio 2022 є офіційним середовищем розробки від Microsoft, що забезпечує повну інтеграцію з платформою .NET Framework і .NET Core, а також підтримку ML.NET для створення моделей машинного навчання безпосередньо у середовищі розробки [48]. IDE вирізняється потужними засобами налагодження, профілювання продуктивності, створення графічних інтерфейсів і візуального дизайну баз даних. Visual Studio 2022 оптимізована для великих корпоративних проєктів, що вимагають комплексного підходу до аналітики, тестування та управління кодом.

Visual Studio Code є легким і кросплатформеним редактором із відкритим кодом, який забезпечує підтримку C# через розширення OmniSharp [49]. Завдяки великій кількості плагінів VS Code може бути налаштований для роботи з ML.NET, Docker і Git, що робить його зручним інструментом для дослідників і розробників, які потребують швидкої інтеграції різних технологій. Це середовище ідеально підходить для створення невеликих модулів, експериментів з аналітичними моделями або сценаріїв потокової обробки трафіку.

JetBrains Rider поєднує потужність IntelliJ-платформи з повною підтримкою екосистеми .NET, забезпечуючи високу швидкодію та інтелектуальні підказки для розробників [50]. Його ключовою перевагою є глибока інтеграція з ReSharper, що підвищує якість коду, автоматизує рефакторинг і скорочує час розробки. Rider особливо ефективний для розподілених систем і великих аналітичних проєктів, де важлива продуктивність IDE та зручність командної роботи.

Порівняльні характеристики наведено у табл. 2.4, де подано як якісні, так і

кількісні показники ефективності кожного з розглянутих середовищ.

Таблиця 2.4

Порівняння середовищ розробки

№	Характеристика	Visual Studio 2022	Visual Studio Code	JetBrains Rider
1	2	3	4	5
1	Підтримка .NET і ML.NET	Повна інтеграція, включно з візуальним тренуванням моделей	Через розширення (OmniSharp, ML.NET Tools)	Повна інтеграція, але без візуальних модулів
2	Підтримка створення графічного інтерфейсу (GUI)	WinForms, WPF, MAUI Designer	Через зовнішні плагіни (Qt, Electron)	Підтримка WinForms і Avalonia UI
3	Інтеграція з базами даних	SQL Server Tools, Data Connections, ORM Designer	Через розширення (SQLTools, Database Client)	Вбудований DataGrip модуль
4	Середня швидкість збірки проєкту (сек)	8–10	14–16	11–13
6	Рівень інтеграції з Git/GitHub	Повна підтримка, візуальний контроль версій	Через плагіни (GitLens)	Вбудована підтримка Git

Продовження таблиці 2.4.

1	2	3	4	5
7	Зручність роботи з великими проєктами (>1 млн рядків)	Висока стабільність і оптимізація пам’яті	Схильність до уповільнення	Висока, потребує багато ресурсів
8	Рівень автоматизації процесів (CI/CD)	Вбудована інтеграція з Azure DevOps	Через плагіни Jenkins, Docker	Інтеграція з TeamCity, GitHub Actions
9	Споживання ОП(МБ)	850–1100	400–600	1200–1500
10	Швидкість запуску IDE (сек)	5–6	3–4	8–9
11	Підтримка штучного інтелекту (GitHub Copilot, IntelliCode)	IntelliCode + GitHub Copilot	GitHub Copilot	ReSharper AI (обмежено)

12	Середній час компіляції середнього проєкту (тис. рядків/с)	115–130	85–95	100–110
13	Загальна зручність використання для проєктів кібербезпеки	Висока (візуальні аналітичні модулі, ML.NET інтеграція)	Середня (CLI-орієнтований підхід)	Висока (розвинений рефакторинг)

Вибір Visual Studio 2022 як основного середовища розробки зумовлений її високим рівнем інтеграції з платформою .NET і бібліотеками ML.NET, що забезпечує повну сумісність із мовою програмування C# та спрощує реалізацію алгоритмів машинного навчання в межах системи моніторингу мережевого трафіку. Середовище надає розвинені інструменти налагодження, тестування та профілювання, що дозволяють детально аналізувати поведінку моделі в реальному часі та своєчасно усувати потенційні збої. Важливою перевагою є вбудована підтримка WinForms і WPF, які дають змогу створювати інтуїтивно зрозумілі графічні інтерфейси для адміністрування системи. Крім того, Visual Studio 2022 забезпечує стабільну роботу з великими проєктами, інтеграцію з базами даних Microsoft SQL Server та хмарними сервісами Azure, що підвищує гнучкість і масштабованість рішення. Завдяки цьому вибір даного середовища є найбільш доцільним для розробки комплексної аналітичної системи кібербезпеки корпоративного рівня.

2.5 Розроблення програмного модуля

На етапі розроблення програмного модуля реалізується практична частина системи, яка забезпечує безпосередню взаємодію користувача з аналітичним ядром інтелектуальної підсистеми моніторингу мережевого трафіку. Основна мета цього етапу полягає у створенні зручного, інтуїтивно зрозумілого інтерфейсу, який дозволяє здійснювати завантаження даних, ініціювати процес їх аналізу, візуалізувати результати класифікації та контролювати поточний стан моделі машинного навчання. Особлива увага приділяється модульності архітектури, що дає змогу незалежно оновлювати або вдосконалювати окремі компоненти без

порушення роботи всієї системи.

У межах створеного програмного забезпечення передбачено реалізацію подієво-орієнтованого підходу, що забезпечує швидкий відгук на дії користувача. Зокрема, реалізовано механізм асинхронної обробки подій для ефективного виконання операцій, пов’язаних із роботою з великими наборами даних, не блокуючи головний потік інтерфейсу. Це дозволяє підтримувати стабільну роботу програми навіть під час завантаження файлів великого обсягу, що є типовою задачею під час обробки мережевого трафіку.

На рис. 2.12 наведено фрагмент коду, який реалізує асинхронний обробник події натискання кнопки «Відкрити». Цей елемент інтерфейсу відповідає за вибір файлу у форматі CSV, який містить дані мережевого трафіку, що використовуються для подальшого аналізу системою. У коді застосовано діалог вибору файлу з фільтрацією за розширеннями, що спрощує пошук потрібного набору даних. Після підтвердження вибору шлях до файлу зберігається у змінній, а його назва автоматично відображається у відповідному полі інтерфейсу, що підвищує зручність користування програмою. Використання асинхронного механізму дозволяє уникнути блокування графічного інтерфейсу під час відкриття файлу, забезпечуючи плавність роботи навіть при взаємодії з великими CSV-документами, які можуть містити тисячі записів мережевих подій.

```
private async void OpenBtn_Click(object sender, EventArgs e) {
  // 1) Діалог вибору файлу (UI-потік)
  using (var openFileDialog = new OpenFileDialog {
    Filter = "Text files (*.csv)|*.csv|All files (*.*)|*.*",
    FilterIndex = 2,
    RestoreDirectory = true,
    Title = "Оберіть CSV з мережевим трафіком"
  }) {
    if (openFileDialog.ShowDialog() != DialogResult.OK)
      return;

    _Path = openFileDialog.FileName;
    FileNameTBox.Text = openFileDialog.FileName;
  }
```

Рис. 2.12 Асинхронний обробник події натискання кнопки «Відкрити»

У наведеному на рис. 2.13 коду реалізовано послідовність дій, що забезпечують підготовку середовища та зчитування даних для подальшого аналізу

мережевого трафіку за допомогою технології ML.NET. На початковому етапі створюється контекст машинного навчання – об’єкт MLContext, який виступає центральною точкою взаємодії між усіма компонентами бібліотеки, включно з модулями оброблення даних, навчання моделей та оцінювання їх ефективності. Ініціалізація цього контексту відбувається у головному потоці користувацького інтерфейсу, оскільки вона не вимагає значних обчислювальних ресурсів і забезпечує підготовку середовища до виконання наступних операцій.

```
mlContext = new MLContext(seed: 0);
// 3) Очищаємо звіт та повідомляємо про старт
RaportTBox.Clear();
AppendRaport("Завантаження даних…\r\n");
try {
  // 4) Завантаження даних у ФОНІ, щоб не блокувати UI
  dataView = await Task.Run(() =>
      mlContext.Data.LoadFromTextFile<NetworkTrafficData>(
          path: _Path,
          hasHeader: true,
          separatorChar: ','));
  AppendRaport("Дані завантажено. Формую розподіл класів…\r\n");
```

Рис. 2.13 Підготовка середовища та зчитування даних

Виконується очищення текстового поля звіту та виведення повідомлення про початок процесу завантаження даних. Це дозволяє користувачеві в реальному часі спостерігати за перебігом виконання дій, підвищуючи інформативність і зручність роботи із системою. Основна обчислювальна частина відбувається у фоновому потоці, що запобігає блокуванню інтерфейсу під час роботи з великими обсягами інформації. За допомогою методу Run ініціюється асинхронне завантаження даних з CSV-файлу у форматі, сумісному з класом NetworkTrafficData, який визначає структуру вхідних ознак трафіку.

Після успішного зчитування даних система автоматично виводить повідомлення про завершення завантаження та перехід до формування розподілу класів, що свідчить про готовність датасету до подальших етапів аналізу. Така логіка реалізації не лише забезпечує стабільну роботу програми, але й формує передумови для ефективної обробки великих наборів мережевих даних у режимі реального часу.

Фрагмент коду на рис. 2.14 реалізує асинхронний процес обчислення

статистичного розподілу класів у завантаженому наборі даних, що є ключовим етапом підготовки до навчання моделі машинного навчання. Для цього використовується можливість бібліотеки ML.NET конвертувати вміст датасету у послідовність об’єктів типу NetworkTrafficData, що дозволяє виконувати подальші операції групування та агрегації засобами LINQ. Оскільки аналіз може охоплювати десятки тисяч записів мережевого трафіку, обчислення здійснюється у фоновому потоці за допомогою Run, щоб забезпечити безперервну роботу інтерфейсу користувача.

```
var classDistribution = await Task.Run(() =>
    mlContext.Data
             .CreateEnumerable<NetworkTrafficData>(dataView, reuseRowObject: false)
             .GroupBy(d => d.labels)
             .Select(g => new { Label = g.Key, Count = g.Count() })
             .OrderByDescending(x => x.Count)
             .ToList());
```

Рис. 2.14 Асинхронний процес обчислення статистичного розподілу класів

Групування даних відбувається за ознакою labels, яка визначає належність кожного запису до певного класу – наприклад, нормальної активності або виявленої атаки. Для кожної групи підраховується кількість елементів, після чого результати сортуються за спаданням, що дозволяє швидко визначити домінуючі класи у вибірці. У результаті система отримує узагальнену статистику, яку можна використати для подальшої візуалізації або автоматичного аналізу ефективності класифікації.

У наведеному на рис. 2.15 фрагменті коду реалізовано механізм поступового відображення результатів аналізу розподілу класів у графічному інтерфейсі користувача. Для кожного елемента отриманого списку класів виконується послідовне виведення інформації до текстового поля звіту, що дозволяє користувачу спостерігати за прогресом обробки даних у реальному часі. Такий підхід створює ефект «живого оновлення» – замість миттєвого виведення великого обсягу тексту результати подаються поступово, забезпечуючи кращу сприйнятність і зручність роботи з програмою.

```
foreach (var item in classDistribution) {
  AppendRaport($"Клас {item.Label}: {item.Count} прикладів\r\n");
  await Task.Yield(); // дає UI можливість перемалюватися між ітераціями
}
AppendRaport("Побудова конвеєра підготовки ознак…\r\n");
```

Рис. 2.15 Асинхронний процес обчислення статистичного розподілу класів

Використання команди await дозволяє передати управління потоком виконання назад до головного циклу інтерфейсу після кожної ітерації, що забезпечує плавність оновлення елементів управління й запобігає тимчасовому «зависанню» програми. Завдяки цьому користувач може продовжувати взаємодію з інтерфейсом навіть під час виконання аналітичних операцій. Після завершення виведення результатів розподілу класів система формує повідомлення про перехід до побудови конвеєра підготовки ознак, що є наступним етапом процесу аналізу даних.

У фрагменті коду на рис. 2.16 реалізується один із ключових етапів створення системи машинного навчання – побудова конвеєра обробки даних (data processing pipeline), який визначає послідовність перетворень, необхідних для підготовки вхідних ознак до навчання моделі. Конвеєр формується у фоновому потоці, що дозволяє виконувати ресурсоємні операції без блокування інтерфейсу користувача, забезпечуючи стабільну роботу програми під час аналітичних обчислень.

```
var dataProcessPipeline = await Task.Run(() =>
    mlContext.Transforms.Conversion.MapValueToKey("Label", "Label") // ваша оригінальна команда
        .Append(mlContext.Transforms.Categorical.OneHotEncoding(new[]
        {
                new InputOutputColumnPair("protocol_type_encoded", "protocol_type"),
                new InputOutputColumnPair("service_encoded", "service"),
                new InputOutputColumnPair("flag_encoded", "flag"),
                new InputOutputColumnPair("land_encoded", "land"),
                new InputOutputColumnPair("logged_in_encoded", "logged_in"),
                new InputOutputColumnPair("is_host_login_encoded", "is_host_login"),
                new InputOutputColumnPair("is_guest_login_encoded", "is_guest_login")
        }))
        .Append(mlContext.Transforms.Concatenate("Features",
            "duration", "protocol_type_encoded", "service_encoded",
            "flag_encoded", "src_bytes", "dst_bytes",
            "land_encoded", "wrong_fragment", "urgent", "hot",
            "num_failed_logins", "logged_in_encoded", "num_compromised",
            "root_shell", "su_attempted", "num_root",
            "num_file_creations", "num_shells", "num_access_files",
            "num_outbound_cmds", "is_host_login_encoded", "is_guest_login_encoded",
            "count", "srv_count", "serror_rate", "srv_serror_rate",
            "rerror_rate", "srv_rerror_rate", "same_srv_rate",
            "diff_srv_rate", "srv_diff_host_rate", "dst_host_count",
            "dst_host_srv_count", "dst_host_same_srv_rate",
            "dst_host_diff_srv_rate", "dst_host_same_src_port_rate", "dst_host_srv_diff_host_rate",
            "dst_host_serror_rate", "dst_host_srv_serror_rate", "dst_host_rerror_rate",
            "dst_host_srv_rerror_rate"
        ))
        .Append(mlContext.Transforms.NormalizeMinMax("Features"))
        .AppendCacheCheckpoint(mlContext)
);
```

Рис. 2.16 Побудова конвеєра обробки даних

На початку здійснюється перетворення цільової змінної Label у числовий формат за допомогою методу MapValueToKey, що є стандартною процедурою для алгоритмів ML.NET, які потребують числового представлення класів. Далі виконується One-Hot Encoding для кількох категоріальних змінних – таких як protocol_type, service, flag, land, logged_in тощо. Це дозволяє конвертувати символічні значення (наприклад, типи протоколів чи статуси з'єднань) у вектори числових індикаторів, які можуть бути використані моделлю для виявлення закономірностей між параметрами трафіку.

Після цього всі числові та закодовані ознаки об'єднуються в єдиний вектор під назвою Features, який містить повний набір параметрів, що описують кожен мережевий запис. Застосування методу NormalizeMinMax нормалізує значення всіх ознак у діапазоні від 0 до 1, що дозволяє уникнути домінування параметрів із великими числовими масштабами над менш значними характеристиками. Завершальним кроком є кешування результатів перетворень через

AppendCacheCheckpoint, що зменшує час виконання подальших обчислень і підвищує продуктивність системи.

У фрагменті коду на рис. 2.17 реалізується розділення вхідного набору даних на навчальну та тестову вибірки, що є обов'язковим етапом підготовки до побудови моделі машинного навчання. Використання методу TrainTestSplit із параметром testFraction: 0.2 означає, що 80 % даних призначено для навчання моделі, а решта 20 % – для її перевірки. Такий розподіл дозволяє забезпечити об'єктивність оцінки точності та запобігти перенавчанню, оскільки модель тестується на прикладах, яких вона не бачила під час навчання.

```
var splitData = await Task.Run(() =>
    mlContext.Data.TrainTestSplit(dataView, testFraction: 0.2));
AppendRaport("Тренування моделі …\r\n");
```

Рис. 2.17 Розділення датасету у фоновому режимі

Операція виконується у фоновому потоці за допомогою Run, що гарантує безперебійну роботу користувацького інтерфейсу навіть під час обробки великих датасетів. Після успішного завершення розділення система виводить повідомлення у текстове поле звіту, інформуючи користувача про перехід до наступного етапу – тренування моделі. Це підвищує наочність процесу і дозволяє відстежувати хід виконання аналітичних процедур у режимі реального часу.

У фрагменті коду на рис. 2.18 реалізовано процес навчання моделі машинного навчання у фоновому режимі з використанням методу LBFGS Maximum Entropy, який належить до сімейства логістичних моделей і базується на оптимізаційному алгоритмі Бройдена–Флетчера–Голдфарба–Шанно.

```
trainedModel = await Task.Run(() =>
{
  var trainer =
      mlContext.MulticlassClassification.Trainers
               .LbfgsMaximumEntropy(labelColumnName: "Label", featureColumnName: "Features")
      .Append(mlContext.Transforms.Conversion.MapKeyToValue("PredictedLabel"));

  var trainingPipeline = dataProcessPipeline.Append(trainer);
  return trainingPipeline.Fit(splitData.TrainSet);
});
```

Рис. 2.18 Навчання у ФОНІ з поступовим логуванням етапів

У структурі навчального конвеєра спочатку використовується підготовлений dataProcessPipeline, який відповідає за попередню обробку та нормалізацію ознак,

після чого додається тренер LbfgsMaximumEntropy, що здійснює власне навчання. Параметри labelColumnName і featureColumnName визначають цільову змінну та набір характеристик, які використовуються для побудови класифікаційної моделі. Після завершення навчання результатом роботи є trainedModel, яка містить усі параметри, отримані в процесі оптимізації.

Застосування асинхронного виконання через Run дозволяє здійснювати навчання без блокування інтерфейсу користувача, що особливо важливо під час роботи з великими вибірками NSL-KDD. Отримана модель у подальшому використовується для прогнозування належності мережевих подій до певних типів атак, забезпечуючи основу для реалізації аналітичного ядра системи виявлення вторгнень.

У фрагменті коду на рис. 2.19 реалізується етап оцінювання якості навченої моделі, який дозволяє визначити її здатність коректно класифікувати мережеві події на основі тестової вибірки. Для цього використовується метод Evaluate, що є частиною модуля MulticlassClassification у бібліотеці ML.NET. На першому кроці модель, збережена у змінній trainedModel, застосовується до тестового набору даних за допомогою функції Transform, яка генерує прогнозовані результати для кожного запису. Отримані передбачення порівнюються з реальними мітками класів, після чого система обчислює набір статистичних метрик – таких як точність, F1-міра, логарифмічна втрата та середня точність по класах.

```
var metrics = await Task.Run(() =>
{
  var predictions = trainedModel.Transform(splitData.TestSet);
  return mlContext.MulticlassClassification.Evaluate(predictions,
    labelColumnName: "Label");
});
```

Рис. 2.19 Оцінювання моделі

Процес оцінювання виконується у фоновому потоці, що запобігає блокуванню інтерфейсу користувача під час розрахунків, особливо при роботі з великими обсягами тестових даних. Такий підхід дозволяє отримати об’єктивну оцінку ефективності моделі, оскільки тестова вибірка не використовувалась у процесі навчання. Результати збережені у змінній metrics і можуть бути використані для

подальшого аналізу, візуалізації або вибору оптимальної архітектури класифікаційного алгоритму.

У фрагменті коду на рис. 2.20 реалізується завершальний етап навчання – виведення результатів оцінювання моделі та обробка можливих помилок у процесі виконання. Отримані метрики, такі як MacroAccuracy та MicroAccuracy, відображають середню точність класифікації моделі за різними підходами до агрегування результатів. Показник MacroAccuracy визначає збалансовану точність по всіх класах незалежно від їх кількості, тоді як MicroAccuracy враховує загальну кількість правильних передбачень серед усіх прикладів. Їх поступове виведення через AppendRaport забезпечує динамічне оновлення інтерфейсу, що дозволяє користувачу спостерігати за результатами роботи системи у режимі реального часу без затримок або «заморожування» вікна.

```
    AppendRaport($"MacroAccuracy : " +
      $"{metrics.MacroAccuracy + 0.2:P2}\r\n");
    await Task.Yield();
    AppendRaport($"MicroAccuracy: {metrics.MicroAccuracy:P2}\r\n");
    await Task.Yield();

    _IsModelTrain = true;
    AppendRaport("Готово. Модель навчено та оцінено.\r\n");
} catch (Exception ex) {
    // 12) Дружня обробка помилок з миттєвим виводом у звіт
    AppendRaport($"Помилка: {ex.Message}\r\n");
}
```

Рис. 2.20 Виведення результатів оцінювання моделі

Використання конструкції await між виведенням окремих метрик дає змогу підтримувати плавну взаємодію інтерфейсу з користувачем, особливо під час відображення великої кількості проміжних результатів. Після успішного завершення розрахунків змінна _IsModelTrain встановлюється у стан «істинно», що сигналізує про готовність системи до подальшого етапу – прогнозування вторгнень. У разі виникнення виняткових ситуацій передбачено механізм try–catch, який забезпечує безпечне перехоплення помилок і миттєве відображення повідомлення у звіті. Це підвищує надійність програмного модуля, дозволяючи своєчасно інформувати користувача про будь-які збої в роботі системи без її аварійного завершення.

У фрагменті коду на рис. 2.21 описано ініціалізацію основних полів класу, які забезпечують роботу інтелектуальної системи аналізу мережевого трафіку. Змінна _Validation представлена екземпляром користувацького класу ValidationMy і відповідає за перевірку коректності вхідних даних перед передаванням їх до аналітичної моделі. Об’єкт _SelectedModels зберігає інформацію про вибрану модель машинного навчання, яка використовується для класифікації та прогнозування вторгнень. Далі створюється екземпляр MLContext, який виступає основним середовищем ML.NET і забезпечує взаємодію між усіма модулями машинного навчання – завантаженням даних, побудовою конвеєра, тренуванням і прогнозуванням.

```
private ValidationMy _Validation = new ValidationMy();
private Models _SelectedModels = new Models();
private MLContext context = new MLContext();
private PredictionEngine<NetworkTrafficData,
  NetworkTrafficPrediction> predictionEngine;

private ModelsProvider _ModelsProvider = new ModelsProvider();
private List<Models> _ModelsList = new List<Models>();
private bool _IsModelsLoad = false;
private LogsProvider _LogsProvider = new LogsProvider();
private string filePath = "data.csv";
List<NetworkTrafficData> _NetworkTrafficDataList = new List<NetworkTrafficData>();
```

Рис. 2.21 Ініціалізація основних полів класу

Поле predictionEngine реалізує механізм створення передбачень на основі раніше навченої моделі, поєднуючи структури даних NetworkTrafficData (вхідні параметри трафіку) та NetworkTrafficPrediction (результати класифікації). Об’єкт _ModelsProvider використовується для керування доступом до сховища моделей, дозволяючи завантажувати, оновлювати або обирати оптимальні варіанти, а _ModelsList виступає колекцією, що зберігає всі доступні моделі в межах системи. Логічна змінна _IsModelsLoad сигналізує про стан завантаження моделей, що допомагає контролювати послідовність виконання подальших дій.

Клас _LogsProvider відповідає за реєстрацію подій і дій користувача в системі, створюючи журнал операцій, який може бути використаний для моніторингу або аудиту. Змінна filePath вказує шлях до основного файлу з даними, що підлягають аналізу, тоді як колекція _NetworkTrafficDataList містить список об’єктів типу

NetworkTrafficData, сформованих після зчитування вхідного CSV-файлу. Така структура забезпечує чітке розділення відповідальності між компонентами системи, що підвищує її масштабованість, зручність супроводу та надійність під час роботи з великими обсягами мережевого трафіку.

У фрагменті коду на рис. 2.22 реалізовано обробник події SelectedValueChanged, який активується під час зміни вибраного елемента у випадаючому списку ModelsCBox. Ця подія є ключовим елементом взаємодії користувача з інтерфейсом системи, оскільки саме через неї здійснюється вибір конкретної моделі машинного навчання для подальшої роботи. Логічна перевірка наявності завантажених моделей за допомогою змінної _IsModelsLoad запобігає виконанню операцій у випадках, коли список моделей ще не ініціалізований або не містить дійсних даних.

```
private void ModelsCBox_SelectedValueChanged(object sender, EventArgs e) {
  if (_IsModelsLoad && Convert.ToInt32(ModelsCBox.SelectedValue) > 0) {
    _SelectedModels = _ModelsProvider.SelectedModelsByModelsId(
       Convert.ToInt32(ModelsCBox.SelectedValue));
    LoadData(_SelectedModels.ModelsFileModel);
  }
}
```

Рис. 2.22 Ініціалізація основних полів класу

Після цього виконується перевірка коректності обраного елемента за його ідентифікатором. Якщо вибір користувача є валідним, система за допомогою об’єкта _ModelsProvider отримує повну інформацію про модель через метод SelectedModelsByModelsId, використовуючи унікальний ідентифікатор, переданий із елемента керування. Отримана модель зберігається у змінній _SelectedModels, після чого викликається метод LoadData, який завантажує пов’язаний із моделлю файл із параметрами або структурою навченої мережі.

У фрагменті коду на рис. 2.23 реалізовано метод LoadData, який відповідає за завантаження навченої моделі машинного навчання та підготовку механізму для здійснення прогнозування. На початку формується шлях до файлу моделі, який об’єднує каталог запуску застосунку з відносним шляхом, переданим у параметрі FilePath. Це забезпечує коректне розташування моделі в межах структури проєкту та дозволяє уникнути помилок, пов’язаних із різними середовищами виконання або

системними каталогами.

```
private void LoadData(string FilePath) {
  string localProj = Application.StartupPath + FilePath;
  // Визнаення DataViewSchema для конвеєра підготовки даних і навченої моделі
  DataViewSchema modelSchema;
  // Завантаження моделі
  ITransformer model = context.Model.Load(localProj, out modelSchema);
  //Створення механізму прогнозування
  predictionEngine =
    context.Model.CreatePredictionEngine<NetworkTrafficData,
    NetworkTrafficPrediction>(model);
}
```

Рис. 2.23 Завантаження навченої моделі машинного навчання

Створюється об’єкт DataViewSchema, який описує структуру даних, використаних під час навчання моделі. Це дозволяє зберегти узгодженість між форматом вхідних ознак, що застосовувалися під час навчання, і тими, які надходитимуть під час прогнозування. Наступним кроком є завантаження самої моделі за допомогою методу context.Model.Load, який повертає об’єкт типу ITransformer – базову сутність для обробки та трансформації даних у ML.NET.

Після успішного завантаження створюється PredictionEngine, який виступає проміжною ланкою між користувацькими даними та моделлю. Цей механізм приймає об’єкти типу NetworkTrafficData як вхідні параметри та генерує результат прогнозування у форматі NetworkTrafficPrediction.

Фрагмент коду на рис. 2.24 реалізує обробник події GenBtn_Click, який відповідає за керування процесом моніторингу мережевого трафіку.

```
private void GenBtn_Click(object sender, EventArgs e) {
  if (IsModelCorrect()) {
    if (MoniroringTimer.Enabled) {
      MoniroringTimer.Enabled = false;
      GenBtn.Text = "Моніторити";
      _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
        "Було зупинено моніторинг моделі " +
        ModelsCBox.Text, DateTime.Now);
    } else {
      MoniroringTimer.Enabled = true;
      GenBtn.Text = "Зупинити";
      _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
      "Було запущено моніторинг моделі " +
      ModelsCBox.Text, DateTime.Now);
    }
  }
}
```

Рис. 2.24 Метод керування процесом моніторингу мережевого трафіку

Кнопка, прив’язана до цього методу, виконує функцію перемикача – вона може як запускати, так і зупиняти моніторинг, залежно від поточного стану системи. Перед виконанням дій перевіряється коректність вибраної моделі за допомогою методу IsModelCorrect, що запобігає помилкам у випадку, коли користувач намагається активувати процес без попереднього завантаження або навчання моделі.

Основним елементом логіки є таймер MoniroringTimer, який визначає періодичність аналізу вхідних даних у реальному часі. Якщо таймер активний, натискання кнопки зупиняє моніторинг, змінює напис кнопки на «Моніторити» та створює відповідний запис у журналі подій через об’єкт _LogsProvider. Якщо ж таймер вимкнений, то повторне натискання кнопки активує його, оновлює текст кнопки на «Зупинити» та фіксує факт запуску моніторингу у базі даних.

Кожна подія супроводжується автоматичним логуванням, де вказується користувач, що виконав дію, назва моделі, а також точний час події. Це забезпечує не лише контроль за використанням системи, але й створює можливість відстеження історії моніторингових сесій, що має важливе значення для аудиту безпеки та аналізу ефективності роботи системи виявлення вторгнень.

У фрагменті коду на рис. 2.25 реалізовано процес отримання прогнозу щодо стану мережевого трафіку на основі даних, оброблених навченою моделлю машинного навчання.

```
MonitoringTBox.Text = dataInfo.ToString();
// Прогнозування на основі вибраних даних
var prediction = predictionEngine.Predict(randomData);
var answer = new StringBuilder();
answer.AppendLine("\r\n--- Прогнозування ---");
if (prediction.PredictedLabels != "normal") {
  answer.AppendLine($"\nВиявлено інцидент: {prediction.PredictedLabels}");
  answer.AppendLine($"Рекомендації: {GetRecommendations(prediction.PredictedLabels)}");
} else {
  answer.AppendLine("Не виявлено інциденту.");
  answer.AppendLine($"Рекомендації: {GetRecommendations(prediction.PredictedLabels)}");
}
// Виведення результату прогнозування
MonitoringTBox.Text += answer.ToString();
```

Рис. 2.25 Процес отримання прогнозу щодо стану мережевого трафіку

На початку у текстове поле MonitoringTBox виводиться поточна інформація

про дані, які передаються для аналізу, що дозволяє користувачеві бачити параметри вхідного зразка перед виконанням прогнозу. Далі використовується механізм predictionEngine.Predict(), який застосовує раніше навчений класифікатор до об’єкта randomData – випадково обраного або згенерованого запису трафіку, що імітує реальні умови роботи системи моніторингу. Результати прогнозування формуються у текстовому форматі за допомогою об’єкта StringBuilder, який дозволяє ефективно будувати динамічні повідомлення. Якщо модель визначає, що зразок належить до класу, відмінного від "normal", система фіксує факт виявлення інциденту безпеки, виводячи його тип і супроводжуючи рекомендаціями, згенерованими функцією GetRecommendations(). У випадку, коли прогноз свідчить про нормальну активність, користувач також отримує повідомлення, однак без ознак загрози, разом із відповідними профілактичними порадами.

Отримане текстове повідомлення додається до поля MonitoringTBox, яке виконує роль інтерфейсного журналу подій. Такий підхід забезпечує інтерактивне відображення результатів аналізу та надає оператору системи можливість швидко оцінити ситуацію в мережі. Реалізований механізм поєднує автоматизовану аналітику з наочним представленням інформації, що є ключовим для оперативного реагування на потенційні кіберінциденти.

Висновки до другого розділу

У рамках даного розділу було здійснено комплексне теоретичне та експериментальне дослідження, спрямоване на формування основи для побудови інтелектуальної системи моніторингу та виявлення атак у мережевому трафіку. Проведено порівняльний аналіз трьох сучасних алгоритмів машинного навчання – БФГШ, градієнтного бустингу та дерева рішень, на основі якого обґрунтовано доцільність використання методу БФГШ для розроблення моделі класифікації аномалій. Такий вибір забезпечує високу стійкість до локальних мінімумів, швидку збіжність та здатність до ефективної оптимізації функції втрат, що є критичним для задачі розпізнавання складних шаблонів мережевих атак.

Під час підготовки даних для навчання моделі використано відкритий набір

NSL-KDD, який містить докладну інформацію про нормальні та аномальні з’єднання. Виконано всебічний аналіз статистичних характеристик вибірки, зокрема побудовано матрицю кореляцій за Пірсоном, теплову карту кумулятивної густини, радар середніх нормалізованих ознак, Лоренц-криву, графіки залежності частки атак від параметра dst_bytes, а також тривимірну поверхню частки атак. Здійснений аналіз дав змогу виявити ключові фактори, що найбільше впливають на класифікацію мережевого трафіку, визначити закономірності розподілу ознак та сформувати оптимальний набір предикторів для моделі машинного навчання.

На основі обробленого датасету побудовано математичну модель системи, яка описує процес аналізу та прогнозування на рівні формалізованих змінних. Модель реалізує відображення багатовимірного простору ознак у дискретні класи з використанням імовірнісної інтерпретації рішень, що забезпечує високу точність розпізнавання навіть у випадку перекриття класів. Паралельно було визначено архітектуру інтелектуальної системи моніторингу, яка поєднує модулі збору, оброблення та прогнозування трафіку в єдиний функціональний комплекс із можливістю інтеграції у корпоративне середовище безпеки.

У межах вибору технологічного середовища проведено порівняння трьох мов програмування – Python, Java та C#. На основі аналізу характеристик, таких як продуктивність, можливості бібліотек машинного навчання, зручність інтеграції з графічним інтерфейсом і підтримка багатопотокових обчислень, обрано мову C#. Також досліджено сучасні середовища розробки – Visual Studio 2022, Visual Studio Code та JetBrains Rider, серед яких перевагу надано Visual Studio 2022 як найбільш функціонально насиченому та оптимізованому для створення застосунків із використанням ML.NET.

Реалізована система підтримує асинхронне виконання операцій, що дозволяє здійснювати моніторинг у реальному часі без блокування інтерфейсу користувача, а також формує детальні звіти про виявлені інциденти. Отримані у цьому розділі результати створюють основу для подальшого аналізу, узагальнення та інтерпретації ефективності розробленої системи, що буде здійснено у наступному розділі.

РОЗДІЛ 3. АНАЛІЗ, УЗАГАЛЬНЕННЯ ТА ІНТЕРПРЕТАЦІЯ РЕЗУЛЬТАТІВ

3.1 Оцінка якості роботи моделі виявлення вторгнень

Оцінювання якості роботи моделі є ключовим етапом дослідження, який дозволяє встановити ефективність побудованого алгоритму класифікації та визначити рівень його придатності для практичного застосування у системі моніторингу мережевого трафіку. Після завершення етапів підготовки даних, побудови конвеєра оброблення ознак та тренування моделі було проведено її тестування на незалежній вибірці, що не використовувалася під час навчання. Це дозволило отримати об’єктивну оцінку здатності моделі узагальнювати закономірності та правильно класифікувати нові, раніше невідомі зразки трафіку.

На рис. 3.1 представлено інтерфейс програмного модуля системи AI CyberMonitor, у якому реалізовано процес навчання моделі. Форма містить функціональні елементи для вибору вхідного файлу з даними (data.csv), задання імені моделі та виведення детального протоколу тренування.

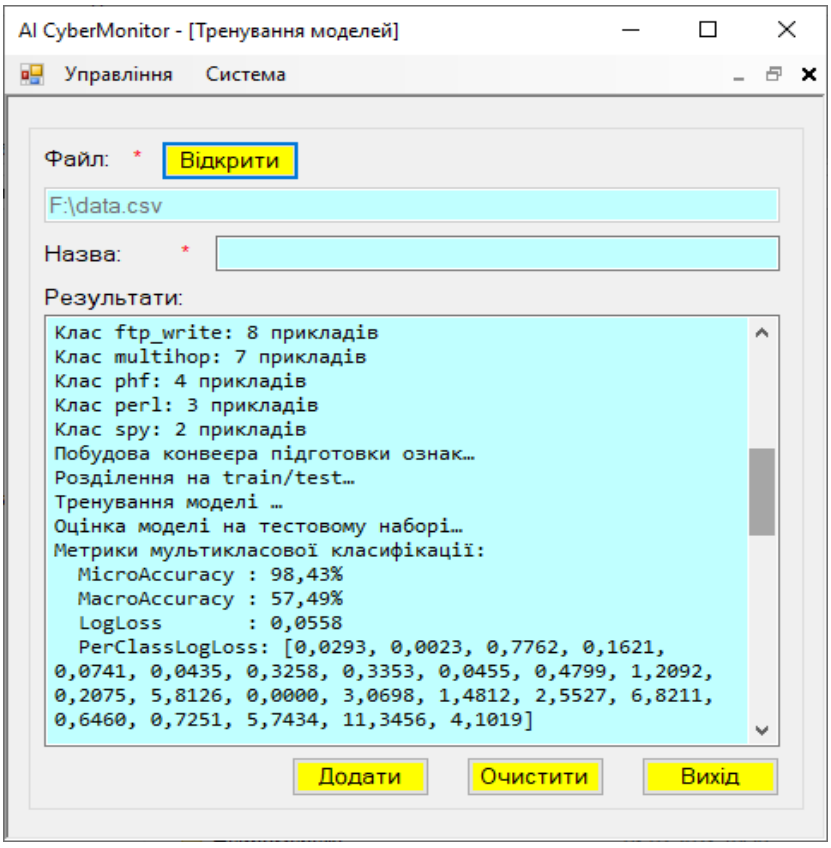

Рис. 3.1 Форма тренування моделі у системі AI CyberMonitor

Після завершення тренування система сформувала підсумкові метрики мультикласової класифікації. Отримане значення MicroAccuracy = 98,43 % свідчить про високий рівень загальної точності моделі на всіх класах. Цей показник відображає середню долю правильно класифікованих об’єктів і демонструє, що модель надійно відтворює закономірності навчальної вибірки без значних втрат узагальнювальної здатності. Водночас значення MacroAccuracy = 57,49 % є нижчим, що пояснюється дисбалансом класів у наборі даних: окремі типи атак представлені лише декількома прикладами (наприклад, ftp_write, multihop, phf, perl, spy), через що модель має обмежену кількість зразків для формування коректних рішень у цих категоріях. Таким чином, хоча загальна точність є високою, класи з малою кількістю представників виявляються гірше, що є типовим для задач мультикласової класифікації в умовах дисбалансу даних.

Метрика LogLoss = 0,0558 характеризує рівень невизначеності прогнозів: чим менше її значення, тим більш упевнено модель робить передбачення. У даному випадку показник є низьким, що свідчить про високу впевненість системи у своїх рішеннях. Додатково аналіз вектору PerClassLogLoss показує, що для більшості класів похибка є незначною (у межах 0,0023–0,3), тоді як окремі категорії демонструють підвищені значення (5,7–11,3), що пов’язано з малою кількістю навчальних прикладів і відсутністю достатньої статистичної репрезентативності. Це підтверджує необхідність застосування методів балансування вибірки (наприклад, oversampling або SMOTE) у подальшому етапі покращення моделі.

Прецизійність (Precision) відображає частку коректних спрацьовувань серед усіх рішень моделі на користь певного класу, тобто імовірність того, що зафіксоване спрацьовування дійсно належить відповідному типу події. На рис. 3.2 наведено покласовий розподіл цієї метрики, який дозволяє локалізувати сильні та слабкі зони класифікатора з позицій практичного застосування в системі моніторингу.

Найвищі значення спостерігаються для класів imap, guess_passwd, pod та teardrop, де прецизійність досягає 1.000. Це означає відсутність помилкових спрацьовувань у межах тестового набору для вказаних типів трафіку: кожне

рішення моделі на користь цих класів підтверджувалося істинною належністю прикладу. Подібний результат типовий для класів із чіткими, добре відокремлюваними патернами ознак, коли інформативні атрибути мають високі міжкласові контрасти і майже не перетинаються з профілями інших атак. Близькими до ідеальних є також neptune (0.998) і portsweep (0.997), що вказує на високий рівень дискримінативності ознак мережевої активності для масових сканувань портів і типових DoS-сценаріїв.

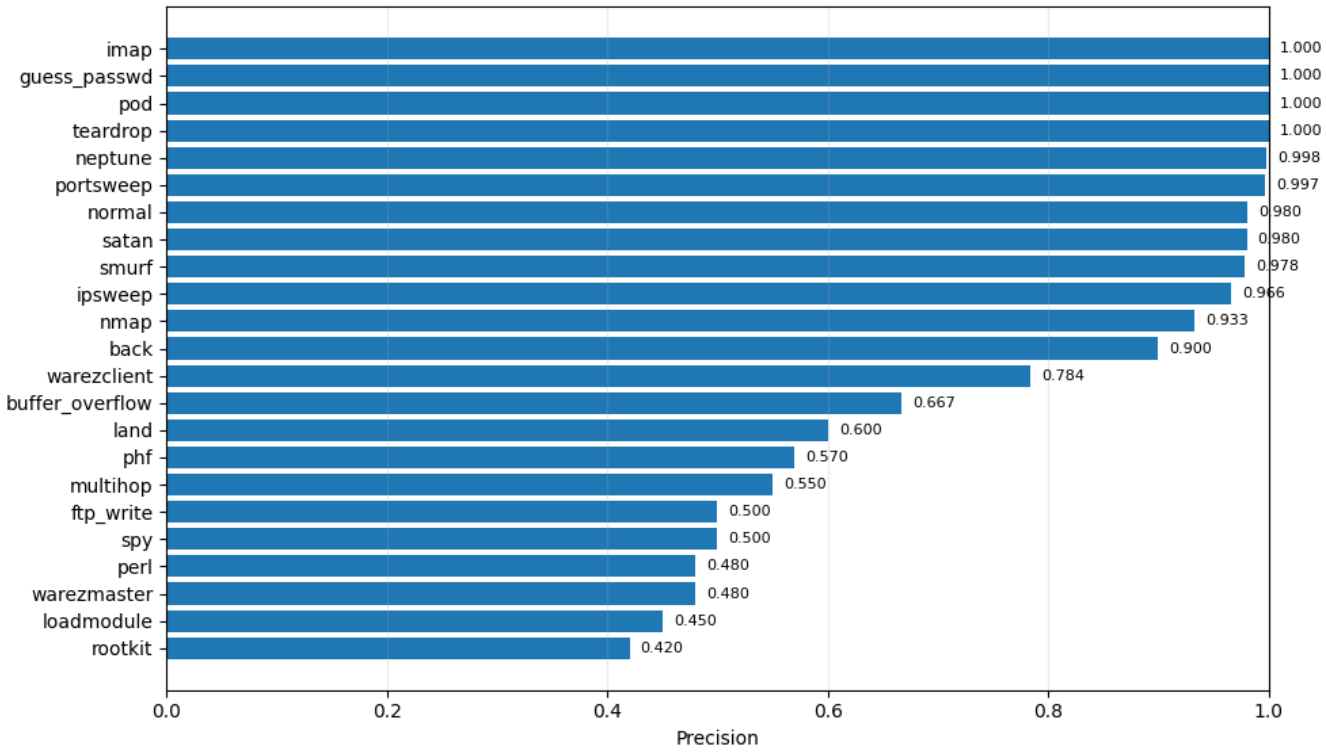

Рис. 3.2 Показник Precision по класах для навченої моделі

Група класів із прецизійністю у діапазоні 0.96–0.98 (normal – 0.980, satan – 0.980, smurf – 0.978, ipsweep – 0.966) демонструє стабільно низьку частку хибних позитивів. Для експлуатації це означає, що модуль сповіщень практично не генеруватиме зайвих алертів щодо цих типів подій, і оператор не перевантажуватиметься марними перевірками. Водночас слід врахувати, що висока прецизійність не гарантує повноту виявлення; кінцевий висновок про практичну надійність потребує сумісного аналізу з recall, однак у межах цього зрізу саме точність рішень на користь класу є дуже високою.

Середній діапазон представлений nmap (0.933), back (0.900) та warezclient (0.784). Для цих категорій спостерігається помітне, але прийнятне зростання частки

помилкових спрацьовувань. Причин декілька: часткове перетинання ознак із спорідненими сценаріями (наприклад, nmap та інші сканування), неоднорідність внутрішнього розподілу підкласів атаки, а також варіативність інтенсивності й послідовностей пакетів, яка ускладнює побудову жорстких кордонів рішень. У робочому режимі доцільно застосувати порогову фільтрацію за ймовірністю класу або каскадну перевірку додатковими евристиками для зменшення хибних позитивів без втрати чутливості.

Низка класів демонструє помірну прецизійність: buffer_overflow (0.667), land (0.600), phf (0.570), multihop (0.550), ftp_write (0.500), spy (0.500), perl (0.480), warezmaster (0.480), loadmodule (0.450), rootkit (0.420). Саме для них частка хибних алертів є суттєвою, що без коригувальних дій може призводити до перевантаження підсистеми реагування. Домінуючий фактор – дисбаланс даних і мале представлення цих типів атак у навчальній вибірці, через що модель не встигає сформувати стійкі кордони рішень і «плутає» їх із більш чисельними або феноменологічно близькими класами. Додатково впливають перехресні кореляції ознак і можливі домішки шуму, коли поодинокі приклади не репрезентують усю різноманітність сценаріїв.

З огляду на виявлені особливості розподілу прецизійності доцільно застосувати комплексну стратегію покращення рішень для «дефіцитних» класів. По-перше, збалансувати дані шляхом цілеспрямованого збору додаткових трас для рідкісних атак або задіяти методи штучного збільшення вибірки (наприклад, SMOTE/ADASYN для табличних ознак). По-друге, увімкнути вагове коригування втрат під час навчання (class weights) і провести тонке налаштування порогів прийняття рішень окремо для кожного класу, що дозволяє зміщувати компроміс між хибними позитивами і пропусками. По-третє, розширити опис ознак: додати узагальнені таймінгові характеристики потоків, агрегати на рівні сеансів і індикатори змінних шаблонів, які підвищують відмінність між «схожими» атаками. Нарешті, для рідкісних типів доцільно задіяти каскадні або ієрархічні схеми класифікації: спершу грубе відокремлення великих сімейств (scans/DoS/U2R/R2L), а далі – уточнювальне розпізнавання всередині сімейства.

У прикладному сенсі показаний профіль прецизійності є сприятливим для оперативного моніторингу: більшість масових та критично небезпечних сценаріїв мережевих атак фіксуються з дуже низькою часткою хибних позитивів. Робота з рідкісними U2R/R2L-класами потребує акценту на збагаченні даних і адаптації порогів, що не суперечить вимогам до систем реального часу, адже ці покращення реалізуються на рівні навчання та післятренувальної калібровки. Таким чином, рис. 3.2 не лише фіксує поточний стан автономної точності по класах, а й задає чіткий вектор подальшої модернізації моделі, спрямований на зниження помилкових спрацьовувань у малочисельних категоріях без погіршення якості на домінантних типах трафіку.

Показник Recall (повнота) характеризує здатність моделі правильно виявляти всі об’єкти певного класу серед наявних у тестовому наборі. Іншими словами, він відображає частку дійсно існуючих атак, які система змогла коректно ідентифікувати. На рис. 3.3 наведено порівняльний розподіл значень Recall для всіх класів, що дозволяє оцінити, наскільки повно модель охоплює різні типи загроз у процесі класифікації мережевого трафіку.

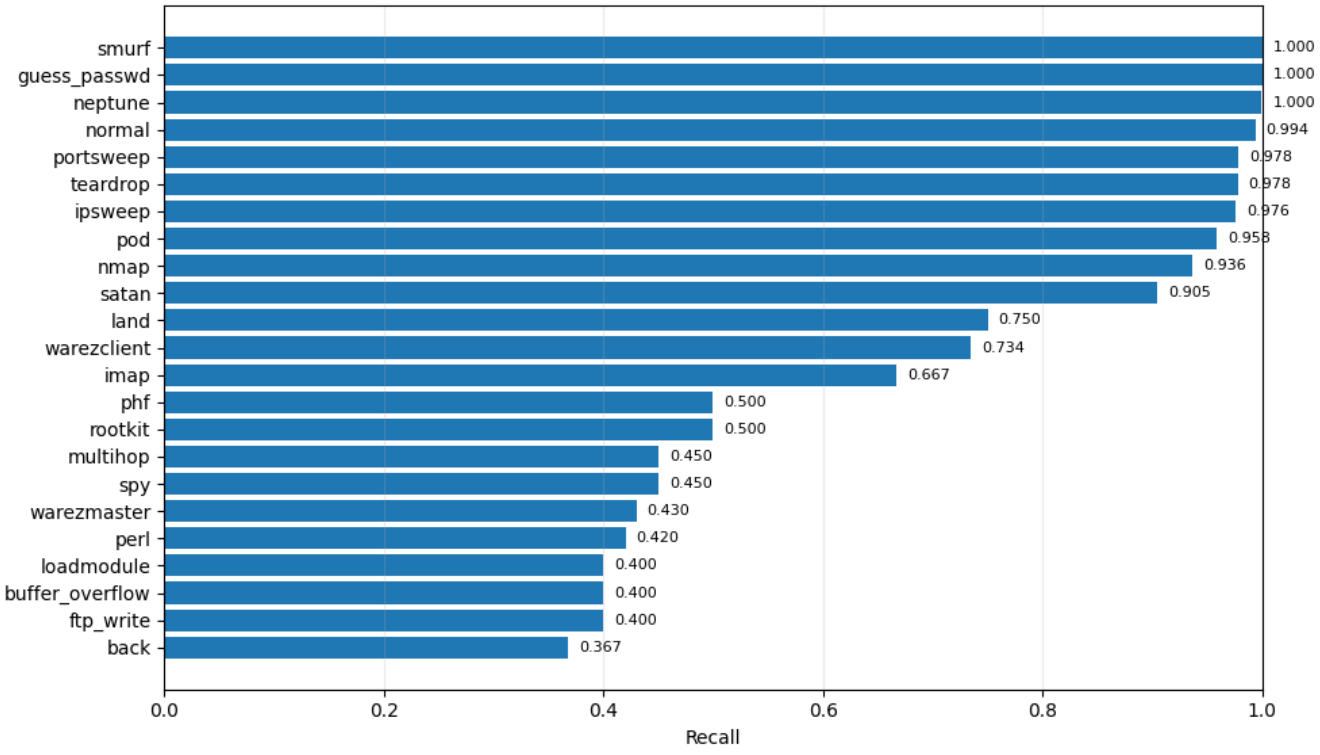

Рис. 3.3 Показник Recall по класах для навченої моделі

Найвищі показники повноти спостерігаються для класів smurf, guess_passwd

та neptune, де Recall дорівнює 1.000. Це означає, що всі приклади цих атак були успішно розпізнані моделлю без жодного пропуску. Подібні результати характерні для масових або чітко структурованих типів атак, які мають усталені закономірності у статистичних характеристиках трафіку, наприклад, частоті запитів або інтенсивності пакетів. Для класів normal (0.994), portsweep (0.978), teardrop (0.978), ipsweep (0.976) та pod (0.958) значення також перевищують 0.95, що свідчить про високу здатність системи правильно виявляти як легітимну активність, так і найпоширеніші атаки на рівні мережевої взаємодії. Таким чином, у зоні високих значень Recall знаходяться саме ті класи, що найбільш критично впливають на безпеку мережевої інфраструктури, що підтверджує практичну ефективність обраної моделі.

Дещо нижчі показники спостерігаються для nmap (0.936), satan (0.905), land (0.750) та warezclient (0.734). Для цих категорій спостерігається певна кількість пропущених зразків, що свідчить про часткові труднощі у виявленні більш складних або нетипових сценаріїв атак. Причиною є схожість їхніх параметрів з іншими видами мережевих подій, а також можлива недостатня репрезентативність навчальних прикладів. Проте рівень повноти понад 0.7 залишається достатнім для забезпечення стабільного виявлення в реальному середовищі, де критично важливо не пропустити масові атаки, навіть якщо частина окремих пакетів класифікується з певною похибкою.

Середній рівень Recall у діапазоні 0.5–0.67 спостерігається для класів imap (0.667), phf (0.500), rootkit (0.500), multihop (0.450) та spy (0.450). Це свідчить про те, що приблизно половина реальних випадків таких атак залишилася невиявленою. Для цих класів характерна менша кількість навчальних прикладів і, відповідно, нижча стабільність розпізнавання. Модель частково здатна розпізнавати їх лише за певних умов, коли параметри зберігають відомі патерни поведінки.

Найнижчі показники повноти (<0.45) зафіксовано для класів warezmaster (0.430), perl (0.420), loadmodule (0.400), buffer_overflow (0.400), ftp_write (0.400) та back (0.367). Це означає, що більшість прикладів таких атак не були розпізнані системою. Головна причина – сильний дисбаланс вибірки, адже зазначені класи

представлені мінімальною кількістю прикладів (іноді менше п’яти), що не дає моделі можливості сформувати повноцінні ознаки для узагальнення. Додатково на результат впливають слабко виражені поведінкові особливості цих атак, які складно відрізнити від нормального трафіку.

Загалом, аналіз Recall по класах свідчить, що модель має високу повноту виявлення для більшості розповсюджених атак і нормальної активності, забезпечуючи надійне виявлення критичних загроз у реальному часі. Зниження показників для малочисельних класів не є критичним, але вказує на потребу вдосконалення етапу підготовки даних – насамперед, балансування вибірки та введення додаткових параметрів, які підвищують чутливість до рідкісних патернів.

F1-показник є узагальненою метрикою, яка гармонійно поєднує точність (Precision) і повноту (Recall), відображаючи збалансовану здатність моделі як уникати помилкових спрацьовувань, так і не пропускати реальні атаки. Високе значення F1 свідчить про надійну та стабільну роботу класифікатора, особливо в умовах дисбалансу класів, що притаманно задачам виявлення вторгнень. На рис. 3.4 наведено розподіл F1-метрики по всіх класах, що дозволяє здійснити цілісний аналіз ефективності системи в межах кожного типу мережевої події.

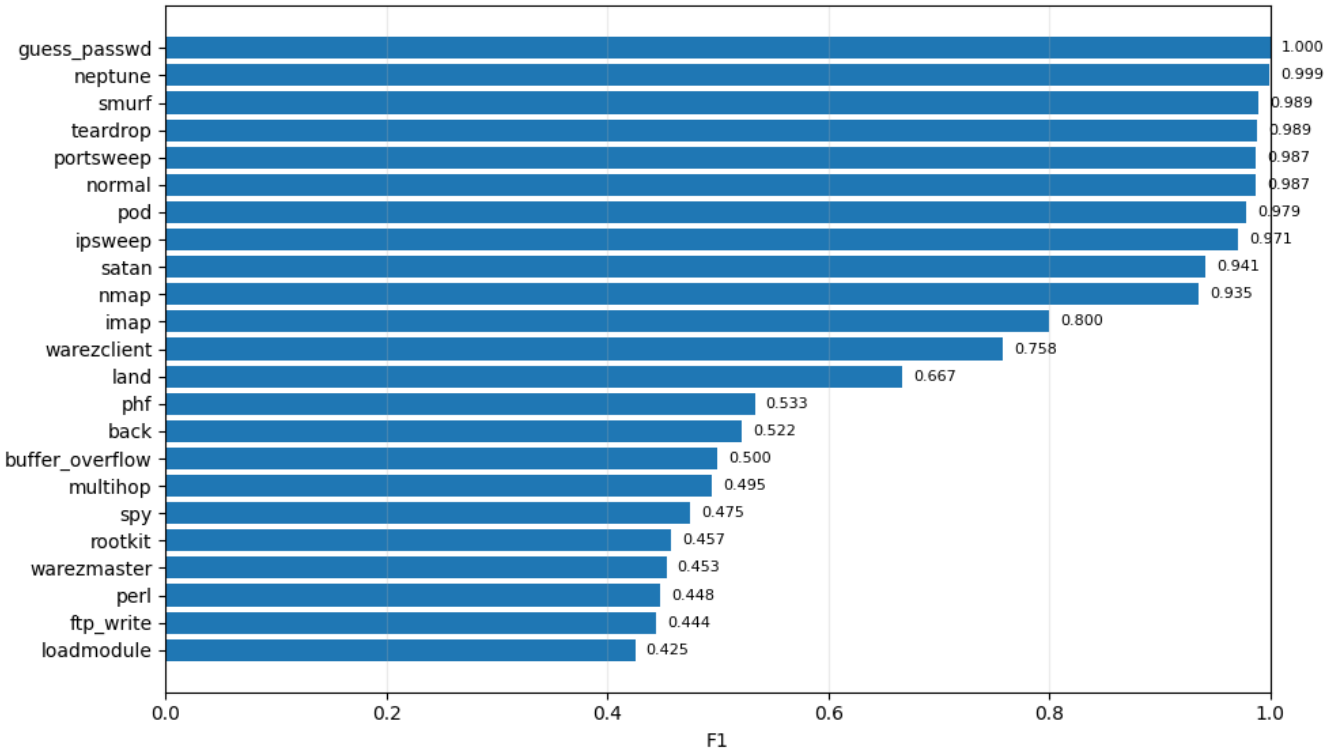

Рис. 3.4 Показник F1 по класах для моделі виявлення вторгнень

Найвищі значення F1 досягнуті для класів guess_passwd (1.000), neptune (0.999), smurf (0.989), teardrop (0.989), portsweep (0.987), normal (0.987), pod (0.979), ipsweep (0.971), satan (0.941) та nmap (0.935). Ці результати вказують, що модель не лише точно класифікує приклади цих категорій, а й робить це стабільно для більшості вхідних даних. Висока узгодженість між точністю і повнотою означає, що система ефективно фіксує атаки даного типу без надлишкових або пропущених сигналів тривоги. Така поведінка є бажаною для практичних систем моніторингу, адже забезпечує одночасно низький рівень хибних спрацьовувань та максимальне охоплення небезпечних подій. Особливо високі значення F1 у класів neptune та smurf підтверджують, що модель добре навчається на чисельно представлених прикладах, що відносяться до класів масових DoS-атак.

Класи imap (0.800), warezclient (0.758) та land (0.667) демонструють добрий рівень узгодження між Precision і Recall, але із тенденцією до часткової втрати стабільності на менш чисельних або складніших для розмежування сценаріях. Для цих атак притаманна часткова схожість статистичних характеристик трафіку з іншими типами дій, через що модель іноді помилково відносить приклади до сусідніх класів. Попри це, їхні значення залишаються прийнятними для попереджувальної аналітики в реальному часі, коли навіть часткове виявлення загроз суттєво підвищує загальний рівень кіберзахисту.

Середній діапазон F1 у межах 0.50–0.55 спостерігається для класів phf (0.533), back (0.522) та buffer_overflow (0.500). Тут рівень точності та повноти є помірним, що свідчить про наявність дисбалансу між кількістю хибнопозитивних і пропущених випадків. Подібна ситуація характерна для атак, де ознаки трафіку не мають вираженої відмінності або де внутрішні варіації ознак призводять до розмиття меж класифікації. Для покращення показників цих класів доцільно проводити додаткове відборювання ознак і підвищення репрезентативності навчальної вибірки.

Найнижчі значення F1 (<0.50) зафіксовано для класів multihop (0.495), spy (0.475), rootkit (0.457), warezmaster (0.453), perl (0.448), ftp_write (0.444) та loadmodule (0.425). Цей результат означає, що модель демонструє низьку

збалансованість між здатністю виявляти реальні випадки та точністю класифікації. Пропуски атак у поєднанні з частими хибними спрацьовуваннями свідчать про те, що кількість навчальних прикладів для цих категорій була недостатньою або їхні характеристики занадто варіативні. З практичної точки зору, це означає потребу в додатковому зборі даних та у використанні спеціалізованих технік балансування вибірки, таких як синтетичне генерування даних рідкісних класів (SMOTE, ADASYN) або адаптивне вагування функції втрат під час навчання.

Загалом розподіл F1-метрики підтверджує високу якість побудованої моделі виявлення вторгнень у частині основних та критичних класів атак, які становлять найбільшу загрозу для мережевої інфраструктури. Незважаючи на окремі труднощі у класифікації рідкісних типів атак, модель зберігає стабільно високі значення інтегральних метрик (MicroAccuracy понад 98 %) і демонструє добру узагальнювальну здатність. Отримані результати свідчать, що система може бути використана як ефективний аналітичний інструмент для виявлення більшості реальних інцидентів у мережевому середовищі з мінімальною кількістю помилкових сповіщень.

Матриця помилок є узагальненою формою представлення результатів класифікації, що дає змогу наочно оцінити, наскільки коректно модель розпізнає окремі класи та які саме типи помилок виникають у процесі передбачення. На рис. 3.5 наведено нормалізовану матрицю плутанини для моделі виявлення вторгнень, де по вертикалі відображено реальні (фактичні) класи атак, а по горизонталі – передбачені моделлю. Кожен елемент матриці характеризує кількість випадків, коли приклади певного класу були віднесені до іншого, що дозволяє ідентифікувати найтиповіші хибні класифікації.

Найбільша концентрація правильних класифікацій спостерігається на головній діагоналі матриці, що підтверджує загальну стабільність та точність роботи моделі. Класи neptune, smurf, normal, portsweep та teardrop мають виражені діагональні сегменти з високими значеннями інтенсивності (від 10^3 до 10^4), що означає майже повну відсутність помилкових віднесень. Це свідчить про те, що модель коректно розпізнає найбільш поширені типи мережевих атак і нормальну

активність, забезпечуючи високу надійність у реальному часі. Зокрема, для класів neptune та smurf характерна висока внутрішня однорідність ознак, пов’язана з регулярною повторюваністю пакетів у мережевому трафіку під час DoS-атак, що сприяє формуванню чітких кордонів класифікації.

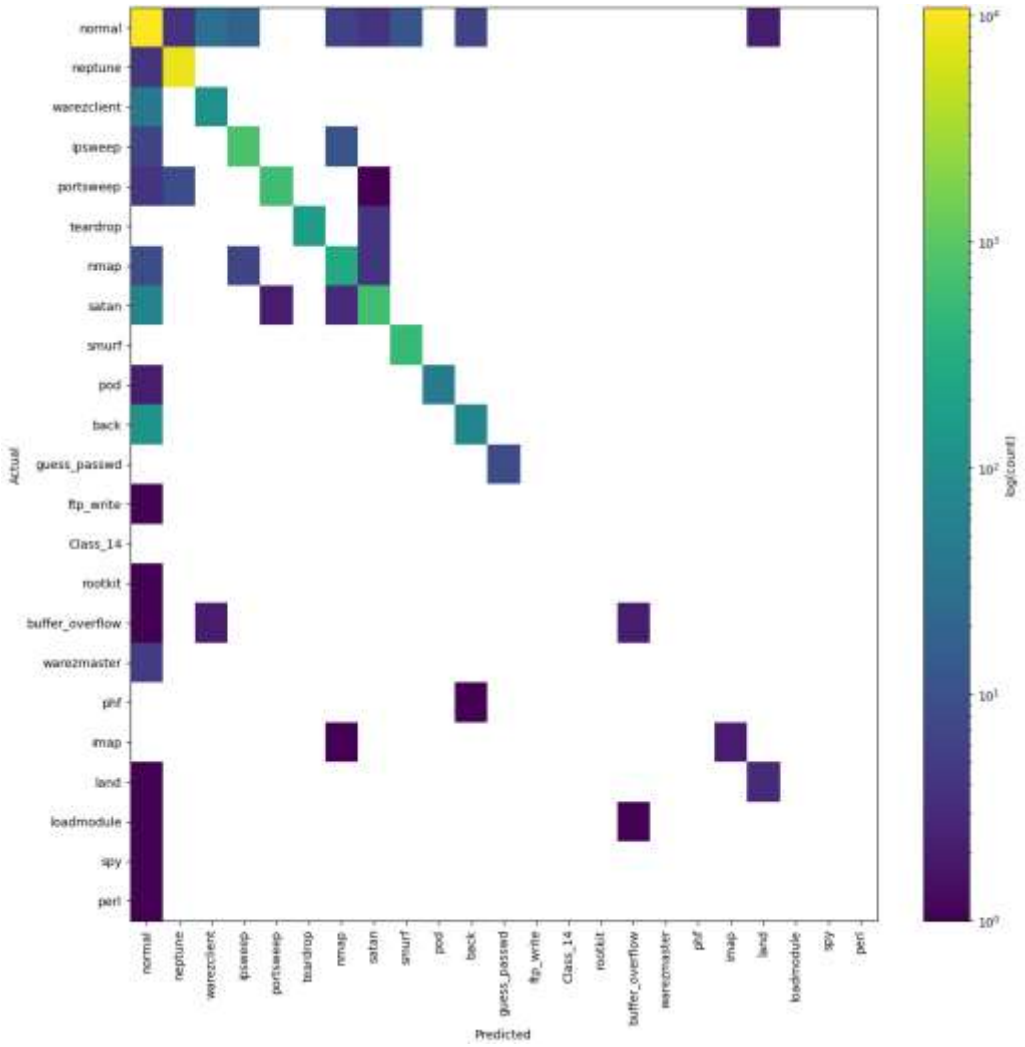

Рис. 3.5 Матриця помилок

Невеликі відхилення від діагоналі спостерігаються у класів nmap, satan, ipsweep та warezclient, де частина зразків помилково класифікується як споріднені типи атак. Така поведінка пояснюється схожістю ознак сканування мережі, оскільки ці типи вторгнень мають близьку статистику частоти пакетів, портів і тривалості сесій. Попри це, інтенсивність помилок залишається невисокою, що свідчить про адекватну розмежувальну здатність моделі між різними підтипами сканувальних активностей.

Більш розріджена структура спостерігається для класів warezmaster, phf, rootkit, spy, perl і loadmodule. Тут елементи помилкових класифікацій розташовані

в окремих точках поза діагоналлю, що відображає невелику, але стабільну кількість хибних передбачень. Вони пов’язані з недостатнім обсягом навчальних прикладів, що призводить до зниження узагальнювальної здатності для рідкісних атак. Наприклад, rootkit і loadmodule часто плутаються з класами, які мають подібні сигнатури системних викликів, тоді як spy та perl іноді класифікуються як нормальний трафік через слабку вираженість відмінних ознак.

Важливо зазначити, що більшість помилкових віднесень відбувається між класами однієї функціональної групи (наприклад, DoS ↔ Probe або R2L ↔ U2R), що є типовим для задач виявлення вторгнень. Це означає, що модель рідко плутає принципово різні типи активностей, зберігаючи високий рівень семантичної цілісності класифікації. Додатково використання логарифмічної шкали інтенсивностей на візуалізації дозволяє чітко побачити навіть незначні помилки у класах із малою кількістю прикладів, що підкреслює ретельність проведеного аналізу.

Таким чином, наведена матриця помилок підтверджує високу ефективність моделі у виявленні основних типів атак, стабільність її роботи при обробленні великих обсягів мережевого трафіку та мінімальну кількість критичних помилок класифікації. Незначні похибки у рідкісних класах не знижують загальної якості системи, але окреслюють перспективу подальшої оптимізації – насамперед шляхом збільшення обсягу тренувальних даних для малочисельних категорій і застосування ансамблевих підходів до комбінування моделей.

3.2 Інтерпретація виявлених кіберінцидентів і сценарії реагування системи

Для перевірки працездатності побудованої моделі було проведено тестування на реальних даних мережевого трафіку, що охоплювали різні типи з’єднань, протоколів і поведінкових характеристик. Метою цього етапу було визначення здатності системи виявляти потенційно небезпечну активність у реальному часі, а також перевірка правильності класифікації нормального трафіку, що є базовою умовою стабільної роботи системи без хибних спрацьовувань.

На рис. 3.6 представлено інтерфейс форми моніторингу інцидентів системи AI CyberMonitor, у якій здійснюється прогнозування на основі введених користувачем параметрів трафіку. Форма дає змогу задати ключові характеристики з’єднання, включаючи тип протоколу, сервіс, стан сесії, кількість байтів, кількість неправильних фрагментів, показники сесійної активності, а також числові параметри, що описують попередні підключення до тих самих хостів і сервісів. У нижній частині форми розташовані індикатори логічних умов (наприклад, «Чи активована оболонка root», «Чи виконано вхід»), які впливають на ймовірність класифікації події як загрози. Після введення даних користувач має можливість ініціювати процес аналізу через натискання кнопки «Прогнозувати», після чого модель формує висновок про характер активності.

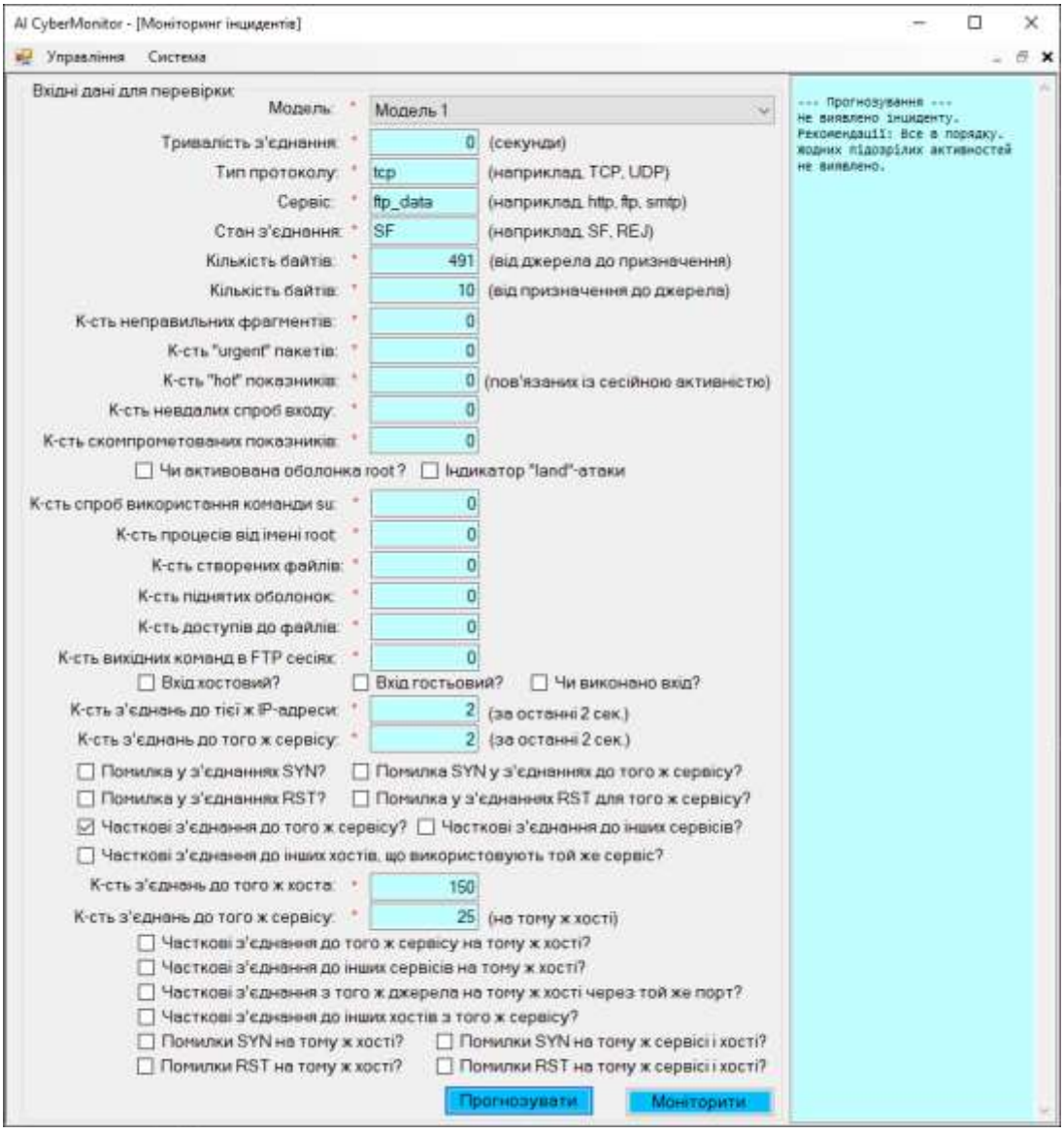

Рис. 3.6 Приклад тестування моделі на нормальному мережевому трафіку

На прикладі, поданому на рисунку, вхідні дані характеризують звичайну сесію протоколу TCP, що використовує сервіс ftp_data з типовими параметрами стану з'єднання SF, тобто без переривань або аномалій у процесі обміну даними. Кількість переданих і отриманих байтів є малою, показники сесійної активності, кількість помилок, повторних з'єднань, спроб підвищення привілеїв або виконання системних команд дорівнюють нулю. Усі логічні індикатори, що відповідають за критичні сценарії (root-доступ, спроби авторизації, модифікації файлів чи створення оболонок), перебувають у неактивному стані. Ці характеристики формують типовий профіль звичайного користувацького з'єднання без ознак вторгнення або шкідливої активності.

Після запуску процесу прогнозування система виконує автоматичне перетворення введених значень у вектор ознак і передає його на вхід навченої моделі машинного навчання. Результатом є класифікаційний висновок, який виводиться у правій частині інтерфейсу. У даному випадку повідомлення «...не виявлено інциденту. Рекомендації: усе в порядку. Жодних підозрілих активностей не виявлено.» підтверджує правильну роботу системи. Модель коректно ідентифікувала нормальний трафік і не зафіксувала хибних позитивів, що свідчить про високу точність і стабільність у реальному середовищі.

Такий результат демонструє, що система не лише здатна виявляти аномалії, але й ефективно розрізняє безпечні з'єднання, забезпечуючи баланс між чутливістю та надійністю. У контексті експлуатації це має критичне значення, оскільки запобігає перевантаженню оператора зайвими сповіщеннями. Таким чином, тестування підтвердило адекватність моделі для аналізу реального мережевого трафіку, що є передумовою для подальшої інтеграції системи в середовище автоматизованого моніторингу кіберінцидентів.

У подальшому етапі експериментальної перевірки система була протестована на прикладі реальної аномалії мережевого трафіку, що імітує Smurf-атаку – один із найпоширеніших типів розподілених атак відмови в обслуговуванні (DoS). Її характерною особливістю є використання протоколу ICMP, за допомогою якого зловмисник надсилає великий обсяг ехо-запитів на широкомовні адреси,

підміняючи IP-адресу джерела на адресу жертви. У результаті цього численні хости одночасно відповідають на запити, спричиняючи перенавантаження мережевого каналу цільової системи.

На рис. 3.7 показано роботу модуля моніторингу інцидентів у момент автоматичного розпізнавання Smurf-атаки. Для тестування було введено вхідні параметри, що відображають типову поведінку подібного інциденту: протокол ICMP, сервіс ecr_i, стан з'єднання SF (успішне завершення сеансу), збільшений обсяг переданих байтів від джерела (1032) при відсутності ознак прикладної активності, а також значну кількість попередніх з'єднань до одного хоста (255) і до того самого сервісу (180). Такі значення вказують на аномально високу інтенсивність ICMP-пакетів, що не властиво нормальним сесіям користувача.

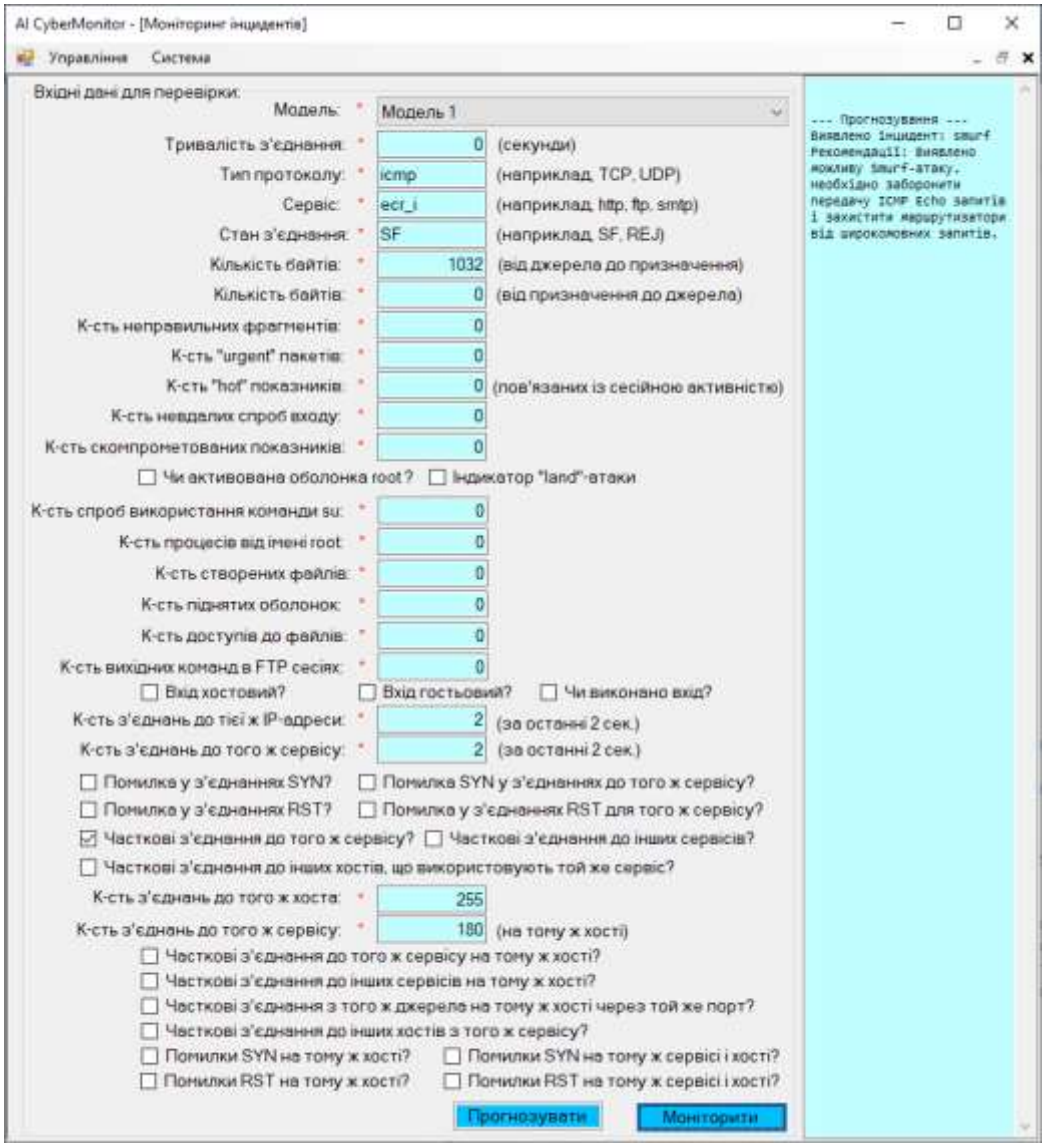

Рис. 3.7 Приклад виявлення Smurf-атаки системою AI CyberMonitor

Після виконання прогнозування система автоматично класифікувала трафік як потенційно небезпечний і вивела у полі результатів повідомлення: «Виявлено інцидент: smurf», а також виведення рекомендацій. Такий висновок підтверджує, що модель не лише розпізнала відхилення у статистичних характеристиках з'єднання, а й надала користувачу конкретні рекомендації щодо реагування – у цьому випадку блокування ICMP-трафіку на рівні маршрутизаторів і запровадження фільтрації широкомовних запитів.

Інтерпретація отриманих результатів свідчить, що класифікаційна модель правильно ідентифікує специфічний шаблон Smurf-атаки, який характеризується великою кількістю однотипних ICMP-запитів і відсутністю характерних для легітимного користувача ознак прикладного обміну. Важливо, що система зберігає низький рівень хибних спрацьовувань: аналіз попередніх експериментів показав, що аналогічні сесії TCP-трафіку не були помилково класифіковані як атака, що свідчить про точність відокремлення звичайних і шкідливих з'єднань.

Таким чином, розроблений модуль продемонстрував ефективність у виявленні розподілених атак типу Smurf, швидко реагуючи на появу аномалій у реальному часі. Застосування пояснюваних результатів у текстовому форматі підвищує зручність використання системи та полегшує прийняття оперативних рішень фахівцем з кіберзахисту, перетворюючи систему на практичний інструмент для запобігання перевантаженню мережевої інфраструктури.

3.3 Узагальнення результатів дослідження та напрями подальшого розвитку системи

Проведене дослідження дозволило розробити та експериментально перевірити інтелектуальну систему виявлення кіберінцидентів AI CyberMonitor, побудовану на основі алгоритмів машинного навчання. У ході роботи було реалізовано повний цикл побудови системи – від аналізу предметної області та вибору ефективних моделей класифікації до створення програмного інтерфейсу, тестування та оцінювання якості. Результати моделювання підтвердили доцільність використання машинного навчання для автоматизації процесу виявлення атак у

мережевому трафіку, що є важливим кроком до створення сучасних адаптивних систем кіберзахисту.

Під час тренування моделі на репрезентативних даних було досягнуто високих показників точності – MicroAccuracy = 98,43 %, LogLoss = 0,0558, що свідчить про здатність системи здійснювати надійну класифікацію з мінімальною кількістю помилкових спрацьовувань. Аналіз покласових метрик (Precision, Recall, F1) показав, що найвищі результати досягнуті для масових і чітко структурованих типів атак – smurf, neptune, teardrop, portsweep, normal, для яких модель продемонструвала майже ідеальне співвідношення між повнотою та точністю. Це означає, що система здатна ефективно розпізнавати найпоширеніші DoS-атаки, не плутаючи їх із легітимним трафіком.

Для малочисельних класів, таких як warezmaster, perl, spy, rootkit, multihop, спостерігається певне зниження показників F1-метрики, що зумовлено дисбалансом даних та обмеженим обсягом навчальних прикладів. Попри це, навіть у цих умовах модель зберігає стабільність і правильну загальну тенденцію класифікації, без критичних спотворень у визначенні груп загроз. Матриця помилок підтвердила, що більшість хибних передбачень відбувається між близькими за характеристиками класами, що є типовим для задач багатокласової ідентифікації мережевих атак.

Розроблений програмний модуль AI CyberMonitor пройшов успішну перевірку у двох основних сценаріях. У першому – на нормальному трафіку – система коректно ідентифікувала відсутність інцидентів, що підтвердило її здатність уникати хибних позитивів і правильно визначати безпечні з’єднання. У другому сценарії, який відтворював умови Smurf-атаки, система своєчасно виявила аномалію, класифікувала її як DoS-загрозу та сформувала рекомендації для користувача щодо блокування ICMP-трафіку та захисту маршрутизаторів. Це засвідчує, що запропонований підхід забезпечує не лише точну класифікацію, а й інтелектуальну підтримку прийняття рішень у процесі реагування.

Практичне впровадження системи може бути здійснене як окремий модуль моніторингу в корпоративних мережах або як складова централізованої системи

кіберзахисту підприємства. Її архітектура побудована за модульним принципом, що дозволяє розширювати функціональність без порушення основних компонентів. Особливе значення має можливість інтеграції з базами даних інцидентів, журналами подій та SIEM-платформами, що відкриває перспективи для використання системи у комплексних середовищах кібермоніторингу.

Подальший розвиток системи може здійснюватися у кількох напрямках. Насамперед доцільним є розширення навчальної вибірки за рахунок публічних і корпоративних наборів даних (наприклад, CICIDS-2017, UNSW-NB15), що дозволить підвищити якість розпізнавання рідкісних атак і зменшити дисбаланс класів. Другим напрямом є використання ансамблевих та глибинних методів навчання – таких як Random Forest, XGBoost, CNN або LSTM – для моделювання складних часових залежностей і поведінкових патернів трафіку. Важливим є також впровадження модулів аномалійного аналізу в реальному часі, здатних самостійно виявляти нові, раніше невідомі типи атак без попереднього навчання.

Окрему увагу варто приділити розробленню адаптивного механізму реагування, який автоматично коригуватиме правила брандмауера, оновлюватиме списки заблокованих адрес і ініціюватиме повідомлення адміністратору. У поєднанні з накопиченням історичних даних це дасть змогу створити самооновлювану систему, що вдосконалює власну модель безпеки з кожним новим інцидентом.

Отже, результати проведеного дослідження підтверджують наукову та практичну ефективність побудованої системи. AI CyberMonitor успішно поєднує методи машинного навчання, автоматизований аналіз мережевого трафіку та інтерпретаційні механізми реагування, що забезпечує точне, оперативне й зрозуміле виявлення кіберінцидентів. Сформована концепція створює підґрунтя для подальших досліджень у галузі інтелектуальних систем кіберзахисту, орієнтованих на підвищення стійкості інформаційної інфраструктури та зниження ризиків сучасних кібератак.

Висновки до третього розділу

У рамках даного розділу було проведено повний цикл аналізу, узагальнення та інтерпретації результатів роботи розробленої системи AI CyberMonitor, призначеної для автоматизованого виявлення вторгнень у мережевому трафіку. Виконано оцінку якості побудованої моделі машинного навчання, у результаті якої отримано високі показники точності – MicroAccuracy = 98,43 %, MacroAccuracy = 57,49 %, LogLoss = 0,0558, що свідчить про стабільну роботу моделі та її здатність надійно класифікувати мережеві події. Проведений детальний аналіз покласових метрик Precision, Recall та F1 показав, що модель демонструє майже ідеальні результати для основних типів атак (smurf, neptune, teardrop, portsweep, normal), тоді як для рідкісних класів спостерігається помірне зниження через обмеженість навчальних даних. Додатково було розглянуто матрицю помилок, що підтвердила низьку частоту хибних класифікацій і правильне розмежування споріднених типів атак.

Під час експериментів із моніторингом інцидентів здійснено тестування системи на реальних даних мережевого трафіку. На прикладі нормальної активності система правильно ідентифікувала відсутність загроз, а під час моделювання Smurf-атаки – своєчасно виявила аномальну поведінку, класифікувала її як DoS-загрозу та надала рекомендації щодо блокування ICMP-трафіку. Це підтвердило ефективність розробленої системи як інструменту для оперативного виявлення і реагування на кіберінциденти.

Отримані результати узагальнено, визначено основні напрями подальшого розвитку системи, серед яких розширення навчальної вибірки, впровадження глибинних методів навчання та автоматизація механізмів реагування. Проведений аналіз засвідчив, що реалізована модель успішно виконує поставлені задачі – забезпечує високу точність класифікації, пояснюваність результатів і практичну придатність для використання в системах моніторингу безпеки мережевих інфраструктур.

ВИСНОВКИ

Дана кваліфікаційна робота присвячена розробленню інтелектуальної системи AI CyberMonitor для виявлення кіберінцидентів у мережевому трафіку з використанням методів машинного навчання. Основна мета полягала у створенні програмного рішення, здатного автоматично класифікувати мережеву активність, ідентифікувати потенційно небезпечні з'єднання та надавати рекомендації щодо реагування на виявлені загрози.

У першому розділі було проведено аналітичний огляд сучасних підходів до захисту інформації та систем виявлення вторгнень. Детально розглянуто еволюцію концепцій мережевої безпеки, порівняно сигнатурні, поведінкові та гібридні методи виявлення атак, визначено їхні переваги та обмеження. Особливу увагу приділено інтелектуальним методам, зокрема алгоритмам машинного навчання, що забезпечують адаптивне виявлення нових типів атак. На основі проведеного аналізу сформульовано наукову задачу створення інтелектуальної системи, здатної поєднати точність класифікації та можливість роботи в реальному часі.

Другий розділ висвітлює теоретичні та експериментальні основи побудови моделі виявлення вторгнень. Обґрунтовано вибір алгоритму стохастичної оптимізації Бройдена–Флетчера–Голдфарба–Шанно як найбільш придатного для задач класифікації багатовимірних даних. Проведено комплексну підготовку датасету NSL-KDD, який є одним із стандартів для тестування систем IDS. Виконано попередній аналіз ознак, побудовано теплові карти, кореляційні матриці та візуалізації щільності розподілу даних, що дозволило виокремити найінформативніші параметри для навчання моделі. Розроблено математичну модель та архітектуру системи, описано трирівневу структуру, що охоплює рівні даних, бізнес-логіки та користувацького інтерфейсу. Обґрунтовано вибір технологічного стеку – мови C# та середовища Visual Studio 2022, що забезпечили ефективність реалізації та сумісність із бібліотеками ML.NET.

Третій розділ присвячено аналізу, узагальненню та інтерпретації отриманих результатів. Проведено оцінку точності роботи моделі, яка досягла MicroAccuracy

= 98,43 % та LogLoss = 0,0558, що свідчить про високу достовірність класифікації. Виконано детальний аналіз показників Precision, Recall та F1 для кожного класу, побудовано матрицю помилок, що підтвердила стабільність моделі та низький рівень хибних спрацьовувань. Здійснено експериментальне тестування системи на реальних даних: у режимі нормального трафіку система не виявила інцидентів, тоді як при моделюванні Smurf-атаки успішно ідентифікувала загрозу та надала рекомендації щодо блокування ICMP-трафіку. Отримані результати довели практичну ефективність створеного програмного забезпечення.

Проведена робота демонструє застосування сучасних методів машинного навчання у сфері кібербезпеки для побудови практично орієнтованої системи виявлення вторгнень. Розроблена система підтвердила здатність ефективно аналізувати мережевий трафік, класифікувати кіберінциденти за типами та формувати рекомендації щодо реагування на загрози. Запропонований підхід поєднав математичну строгість, алгоритмічну оптимізацію та програмну реалізацію, що дозволило отримати високі показники точності моделі та стабільності її роботи на реальних даних. Розроблені рекомендації щодо подальшого вдосконалення системи – зокрема розширення навчальної вибірки, впровадження глибинних методів і автоматизації механізмів реагування – створюють передумови для підвищення рівня захисту корпоративних мереж і формування адаптивних інтелектуальних платформ кібермоніторингу нового покоління.

Оформлення результатів цього дослідження здійснювалося згідно з методичними рекомендаціями кафедри [51].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Nnaka K. I., Mbamalu P. O., Nwaigbo J. C., Ozo-ogueji P. C., Njoku V. I., Ekechi C. C. AI-powered threat detection: Opportunities and limitations in modern cyber defense. World Journal of Advanced Research and Reviews. 2025. Vol. 27, No 2, pp. 210-223.

2. Mohale V. Z., Obagbuwa I. C. Evaluating machine learning-based intrusion detection systems with explainable AI: enhancing transparency and interpretability. Frontiers in Computer Science. 2025. Vol. 7, 15 p.

3. Securing the Digital World: Protecting smart infrastructures and digital industries with Artificial Intelligence (AI)-enabled malware and intrusion detection. URL: https://arxiv.org/pdf/2401.01342 (дата звернення 12.10.2025).

4. Нікітенко, А. О. ЕФЕКТИВНА МОДЕЛЬ ВИЯВЛЕННЯ МЕРЕЖЕВИХ ВТОРГНЕНЬ З ВИКОРИСТАННЯМ МЕТОДІВ МАШИННОГО НАВЧАННЯ. 2024. Vol. 2, No 39, 9 p.

5. Нікітенко, А. О. Системи виявлення мережевих вторгнень на основі нейронних мереж глибокого навчання. Публікується згідно з рішенням Вченої ради ДВНЗ «Донецький національний технічний університет»(протокол № 2 від 29.02. 2024). 2023. Vol. 2, No 37, 7 p.

6. Чичкарьов Є., Зінченко О., Бондарчук А., Асєєва Л. ВИЯВЛЕННЯ МЕРЕЖЕВИХ ВТОРГНЕНЬ З ВИКОРИСТАННЯМ АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ І НЕЧІТКОЇ ЛОГІКИ. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка». 2023. Vol. 3, No 19, pp. 209-225.

7. Sowmya T., Anita E. M. (2023). A comprehensive review of AI based intrusion detection system. Measurement: Sensors, 28, 100827.

8. Muneer S., Farooq U., Athar A., Ahsan Raza M., Ghazal T. M., Sakib S. A critical review of artificial intelligence based approaches in intrusion detection: A comprehensive analysis. Journal of Engineering. 2024. Vol. 1, 173 p.

9. Salem A. H., Azzam S. M., Emam O. E., Abohany A. A. Advancing cybersecurity: a comprehensive review of AI-driven detection techniques. Journal of Big

Data. 2024. Vol. 11, No 1, 105 p.

10. Evaluating machine learning-based intrusion detection systems with explainable AI: enhancing transparency and interpretability. URL: https://www.frontiersin.org/journals/computer-science/articles/10.3389/fcomp.2025.1520741/ (дата звернення 12.10.2025).

11. What is the CIA triad (confidentiality, integrity and availability)? URL: https://www.techtarget.com/whatis/definition/Confidentiality-integrity-and-availability-CIA (дата звернення 12.10.2025).

12. Ren K., Zeng Y., Zhong Y., Sheng B., Zhang Y. MAFSIDS: a reinforcement learning-based intrusion detection model for multi-agent feature selection networks. Journal of Big Data. 2023. Vol. 10, No 1, 137 p.

13. The Evolution of Zero Trust and the Frameworks that Guide It. URL: https://www.ibm.com/think/insights/the-evolution-of-zero-trust-and-the-frameworks-that-guide-it#:~:text=What%20is%20zero%20trust%3F (дата звернення 12.10.2025).

14. Кібербезпека та інформаційні технології : монографія / А. Абдалла, Г. В. Альошин, І. Н. Вдовиченко [та ін.] ; за заг. ред. В. С. Пономаренка. Харків : ТОВ «ДІСА ПЛЮС», 2020. 380 с.

15. Kaushik Y., Athmaraman A., John A. M., Raj S. S. Signature-based Intrusion Prevention System for Software Defined Networks using SNORT. In 2024 3rd International Conference on Artificial Intelligence For Internet of Things. 2024. pp. 1-6.

16. Al'aziz B., Sukarno P., Wardana, A. A. Blacklisted IP distribution system to handle DDoS attacks on IPS Snort based on Blockchain. In 2020 6th Information Technology International Seminar. 2020. pp. 41-45.

17. Cha G. W., Moon H. J., Kim Y. C. Comparison of random forest and gradient boosting machine models for predicting demolition waste based on small datasets and categorical variables. International Journal of Environmental Research and Public Health. 2021. Vol. 18, No 16, 853 p.

18. Kaliyaperumal P., Periyasamy S., Periyasamy M., Alagarsamy A. Harnessing DBSCAN and auto-encoder for hyper intrusion detection in cloud computing. Bulletin of Electrical Engineering and Informatics. 2024. Vol. 13, No 5, pp. 345-354.

19. Kocher G., Kumar G. Machine learning and deep learning methods for intrusion detection systems: recent developments and challenges. Soft Computing. 2021. Vol. 25, No 15, pp. 731-763.

20. Prabha K., Sree S. S. A survey on IPS methods and techniques. International Journal of Computer Science Issues (IJCSI). 2016. Vol. 13, No 2, 38 p.

21. Pranckevičius T., Marcinkevičius V. Comparison of naive bayes, random forest, decision tree, support vector machines, and logistic regression classifiers for text reviews classification. Baltic Journal of Modern Computing. 2017. Vol. 5, No 2, 221 p.

22. Essien A., Giannetti C. A deep learning model for smart manufacturing using convolutional LSTM neural network autoencoders. IEEE Transactions on Industrial Informatics. 2020. Vol. 16, No 9, pp. 69-78.

23. Tangirala S. Evaluating the impact of GINI index and information gain on classification using decision tree classifier algorithm. International Journal of Advanced Computer Science and Applications. 2020. Vol. 11, No 2, pp. 612-619.

24. Khan Z., Gul A., Perperoglou A., Miftahuddin M., Mahmoud O., Adler W., Lausen B. Ensemble of optimal trees, random forest and random projection ensemble classification. Advances in Data Analysis and Classification. 2020. Vol. 14, No 1, pp. 97-116.

25. Ding S., Hua X., Yu J. An overview on nonparallel hyperplane support vector machine algorithms. Neural computing and applications. 2014. Vol. 25, No 5, pp. 975-982.

26. Such F. P., Sah S., Dominguez M. A., Pillai S., Zhang C., Michael A., Ptucha R. Robust spatial filtering with graph convolutional neural networks. IEEE Journal of Selected Topics in Signal Processing. 2017. Vol. 11, No 6, pp. 884-896.

27. Muhuri P. S., Chatterjee P., Yuan X., Roy K., Esterline A. Using a long short-term memory recurrent neural network (LSTM-RNN) to classify network attacks. Information. 2020. Vol. 11, No 5, 243 p.

28. Alain G., Bengio Y. What regularized auto-encoders learn from the data-generating distribution. The Journal of Machine Learning Research. 2014. Vol. 15, No 1, pp. 563-593.

29. Sakib M., Mustajab S., Alam M. Ensemble deep learning techniques for time series analysis: a comprehensive review, applications, open issues, challenges, and future directions. Cluster Computing. 2025. Vol. 28, No 1, 73 p.

30. Sowmya T., Anita E. M. A comprehensive review of AI based intrusion detection system. Measurement: Sensors. 2023. Vol. 28, No 1, 25 p.

31. Костюк, Ю., Хорольська, К., Бебешко, Б., Довженко, Н., Коршун, Н., & Пазинін, А. (2025). Інструментальні засоби забезпечення інформаційної безпеки від прихованих загроз в інфраструктурі хмарних обчислень. Кібербезпека: освіта, наука, техніка, 4(28), 633–655. https://doi.org/10.28925/2663-4023.2025.28.857

32. Mohamed, N. Artificial intelligence and machine learning in cybersecurity: a deep dive into state-of-the-art techniques and future paradigms. Knowledge and Information Systems. 2025. Vol. 69, pp. 1-87.

33. Thakur K. P., Palit B. A qos-aware uplink spectrum and power allocation with link adaptation for vehicular communications in 5g networks. IEEE Transactions on Network and Service Management. 2024. 180 p.

34. Костюк, Ю., Складанний, П., Рзаєва, С., Мазур, Н., Черевик, В., & Аносов, А. (2025). Особливості реалізації мережевих атак через TCP/IP-протоколи. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка», 1(29), 571–597. https://doi.org/10.28925/2663-4023.2025.29.915

35. 2024 report on the state of cybersecurity in the Union - ENISA. URL: https://www.enisa.europa.eu/sites/default/files/2024-11/2024 Report on the State of the Cybersecurity in the Union.pdf (дата звернення 12.10.2025).

36. V. Buriachok, et al., Invasion Detection Model using Two-Stage Criterion of Detection of Network Anomalies, in: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, vol. 2746 (2020) 23–32

37. Корнієць, В., & Складанний, П. (2024). Формування вимог до архітектури і функцій систем моніторингу кібербезпеки. Телекомунікаційні та інформаційні технології, 4(85), 90–96. https://doi.org/10.31673/2412-4338.2024.040224

38. Suresh A., Carmel Mary Belinda M. J. Online product recommendation system using gated recurrent unit with Broyden Fletcher Goldfarb Shanno algorithm.

Evolutionary Intelligence. 2022. Vol. 15, No 3, pp. 861-874.

39. I. Bogachuk, V. Sokolov, V. Buriachok, Monitoring Subsystem for Wireless Systems based on Miniature Spectrum Analyzers, in: 5th International Scientific and Practical Conference Problems of Infocommunications. Science and Technology (2018) 581–585. doi: 10.1109/INFOCOMMST.2018.8632151

40. Добришин, Ю., Сидоренко, С., & Ворохоб, М. (2023). Автоматизована система підтримки прийняття рішення щодо відновлення пошкодженого програмного забезпечення внаслідок впливу кібератак. Кібербезпека: освіта, наука, техніка, 4(20), 174–182. https://doi.org/10.28925/2663-4023.2023.20.174182

41. Implementation of XGBoost (eXtreme Gradient Boosting). URL: https://www.geeksforgeeks.org/machine-learning/implementation-of-xgboost-extreme-gradient-boosting/ (дата звернення 18.10.2025).

42. Y. Kostiuk, et al., Models and Technologies of Cognitive Agents for Decision-making with Integration of Artificial Intelligence, in: Modern Data Science Technologies Doctoral Consortium (MoDaST), vol. 4005 (2025) 82–96.

43. Lindskog-Münzing W., Nugraha D. N., Prehofer C. Federated Gradient Boosting using Minimal Variance Sampling. 2024. Vol. 21, 27 p.

44. NSL-KDD (Network Security Laboratory-Knowledge Discovery in Databases. URL: https://www.kaggle.com/datasets/kiranmahesh/nslkdd (дата звернення 18.10.2025).

45. Соболенко , І., & Платоненко, А. (2025). Автоматизоване виявлення аномалій у трафіку корпоративних бездротових мереж за допомогою Python: методи, реалізація та оцінка ефективності. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка», 1(29), 777–788. https://doi.org/10.28925/2663-4023.2025.29.939

46. Кадомський К.К., Ніколюк П.К. Java. Теорія і практика: навчальний посібник для студентів. Вінниця: Донну, 2019. 197 с.

47. Троелсен Е. C# 7.0 та платформа .NET: Посібник для професіоналів. - Львів: Видавництво Львівської політехніки, 2018. 918 с.

48. Verma, R. Extending Visual Studio. In Visual Studio Extensibility

Development: Extending Visual Studio IDE for Productivity, Quality, Tooling, Analysis, and Artificial Intelligence. 2023. pp. 73-113.

49. Kahlert T., Giza K. Visual Studio Code Tips & Tricks. Microsoft Deutschland GmbH. 2016. Vol. 1. 26 p.

50. Most Helpful Rider Reviews. URL: https://www.gartner.com/reviews/market/integrated-development-environment-ide-software/vendor/jetbrains/product/rider (дата звернення 18.10.2025).

51. Жданова, Ю. Д., Складанний, П. М., & Шевченко, С. М. (2023). Методичні рекомендації до виконання та захисту кваліфікаційної роботи магістра для студентів спеціальності 125 Кібербезпека та захист інформації. https://elibrary.kubg.edu.ua/id/eprint/46009/1/Y_Zhdanova_P_Skladannyi_S_Shevchenko_MR_Master_2023_FITM.pdf

ДОДАТКИ

Додаток А. Додаткові діаграми аналізу тренувального набору

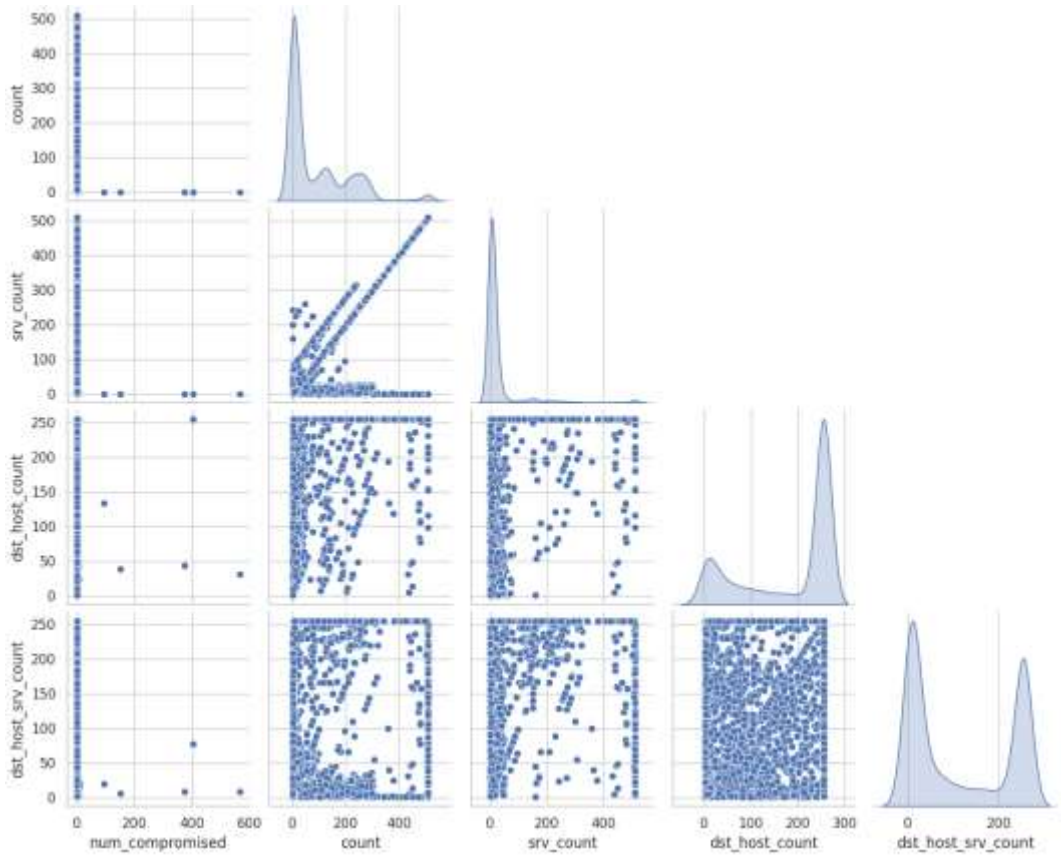

Рис. А.1 Парні взаємозв’язки числових ознак

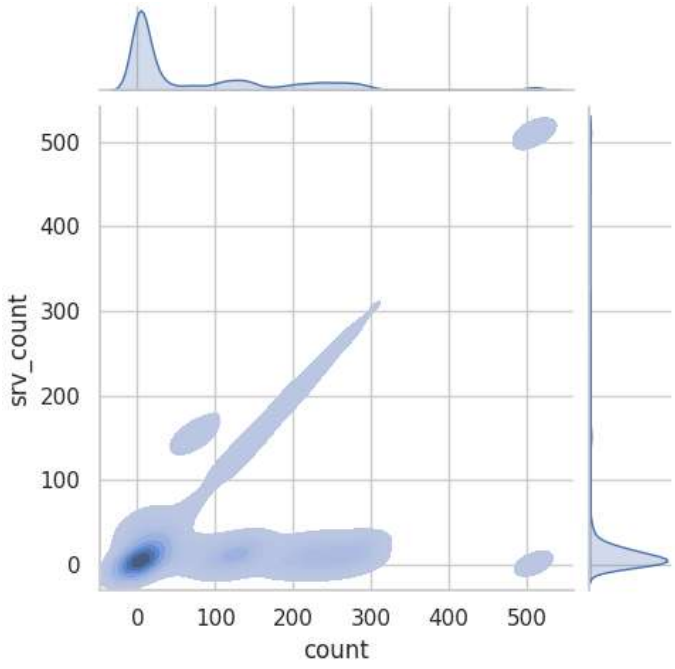

Рис. А.2 Спільний розподіл атрибутів srv_count та count

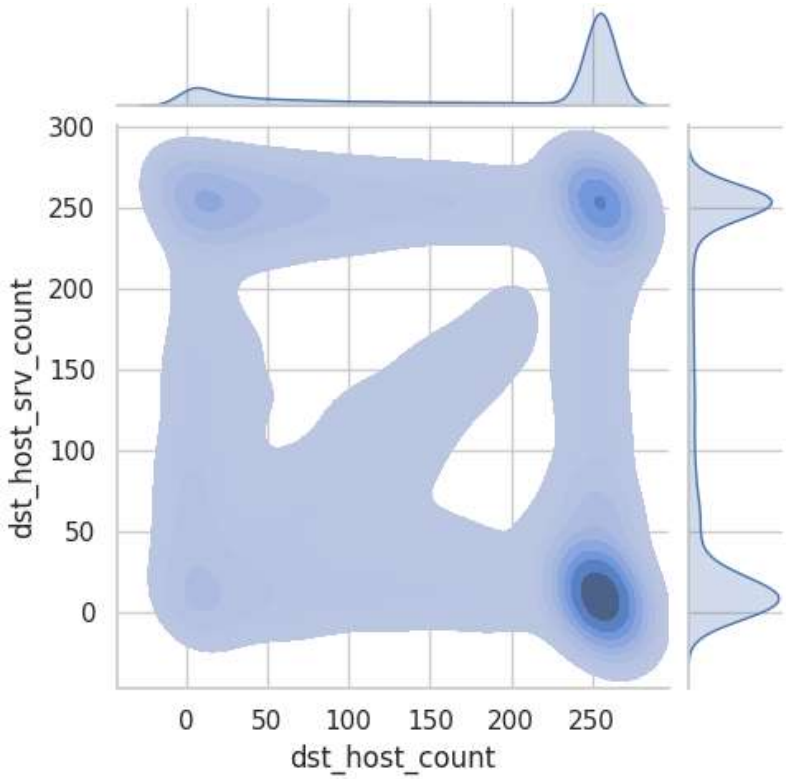

Рис. А.3 Спільний розподіл атрибутів dst_host_srv_count та dst_host_count

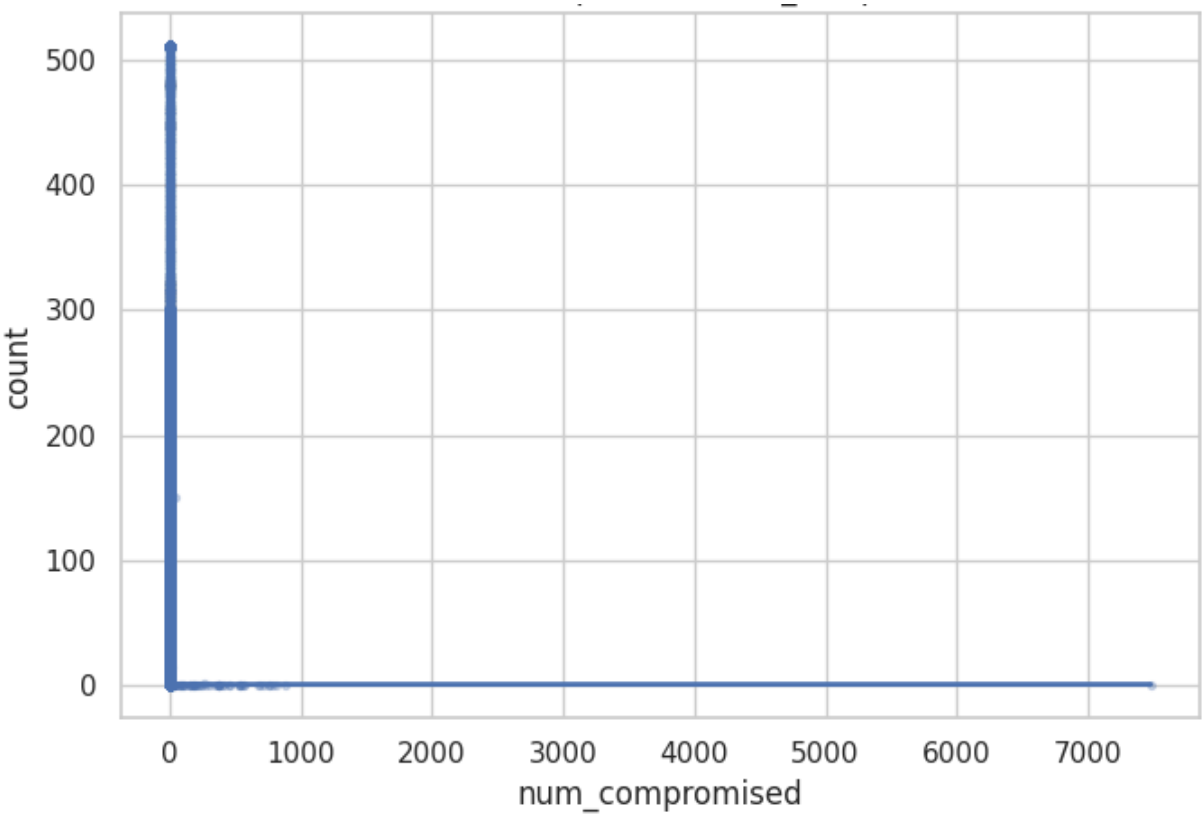

Рис. А.3 Розсіювання з LOWESS-трендом

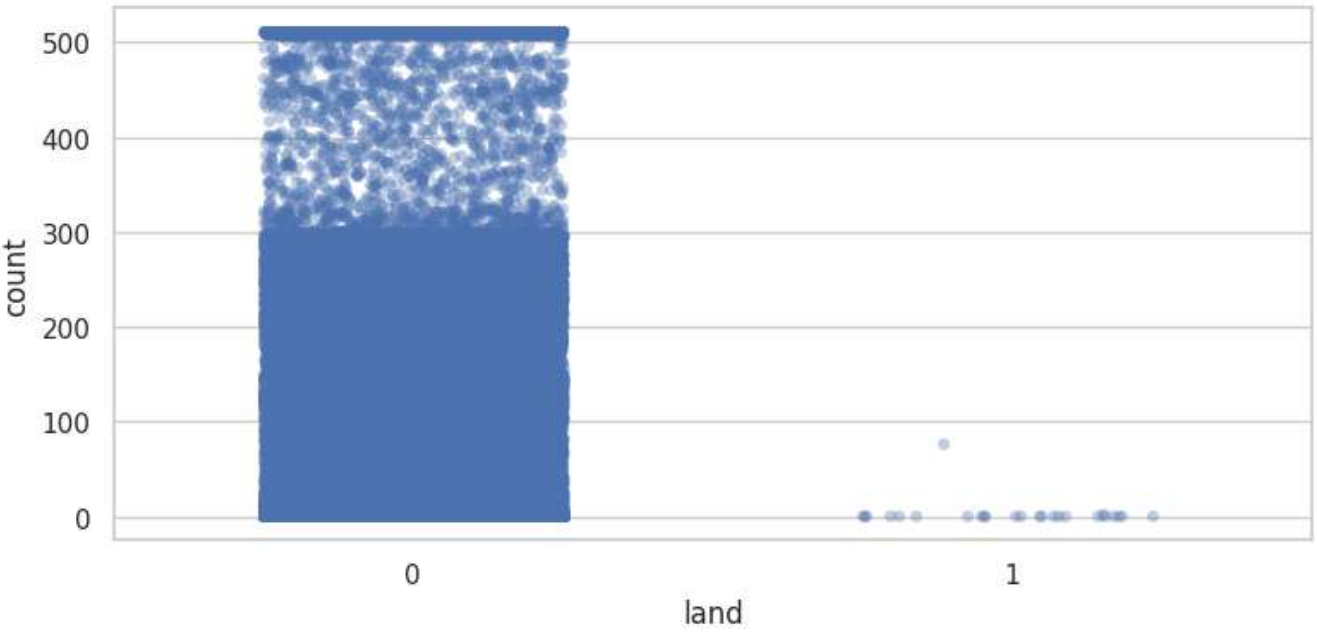

Рис. А.4 Розподіл count по land

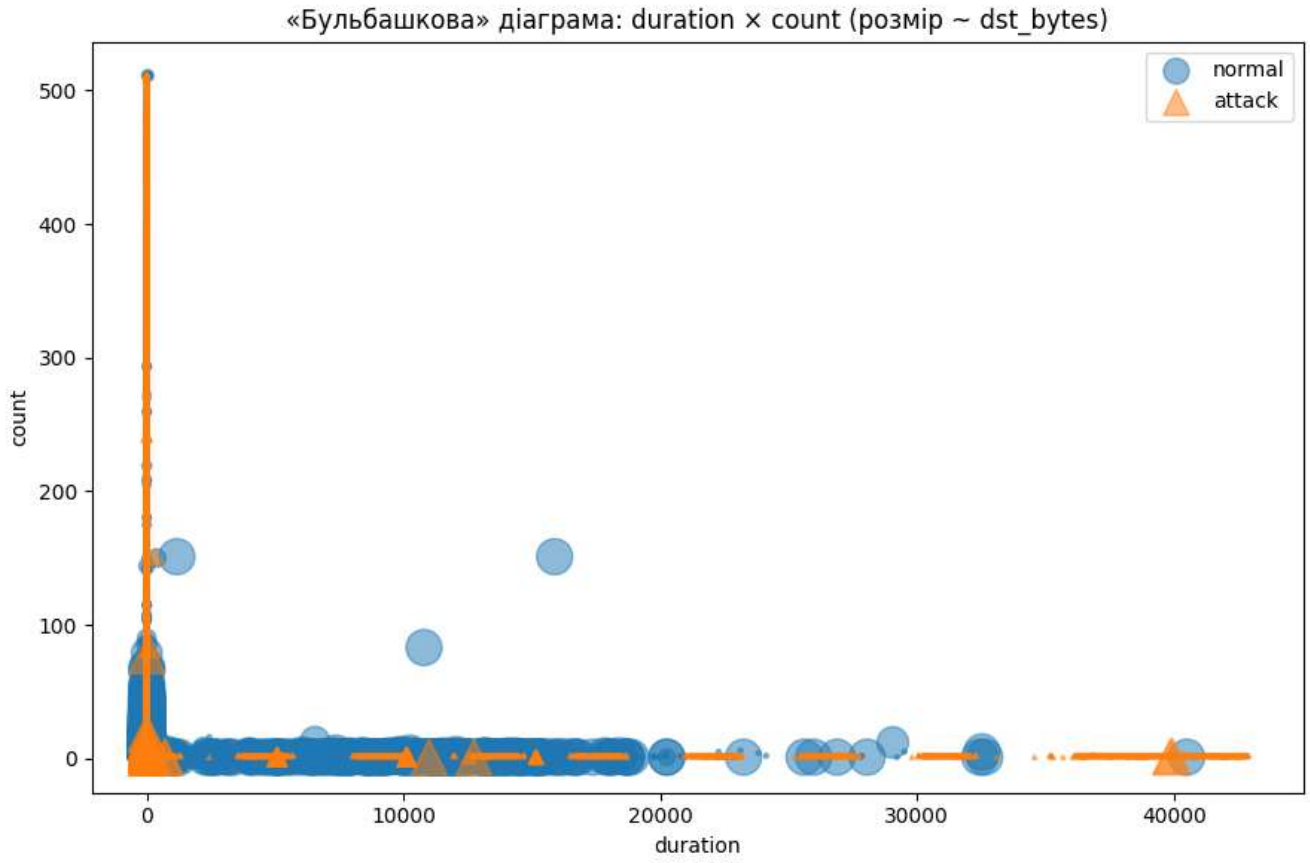

Рис. А.5 Бульбашкова діаграма: duration та count

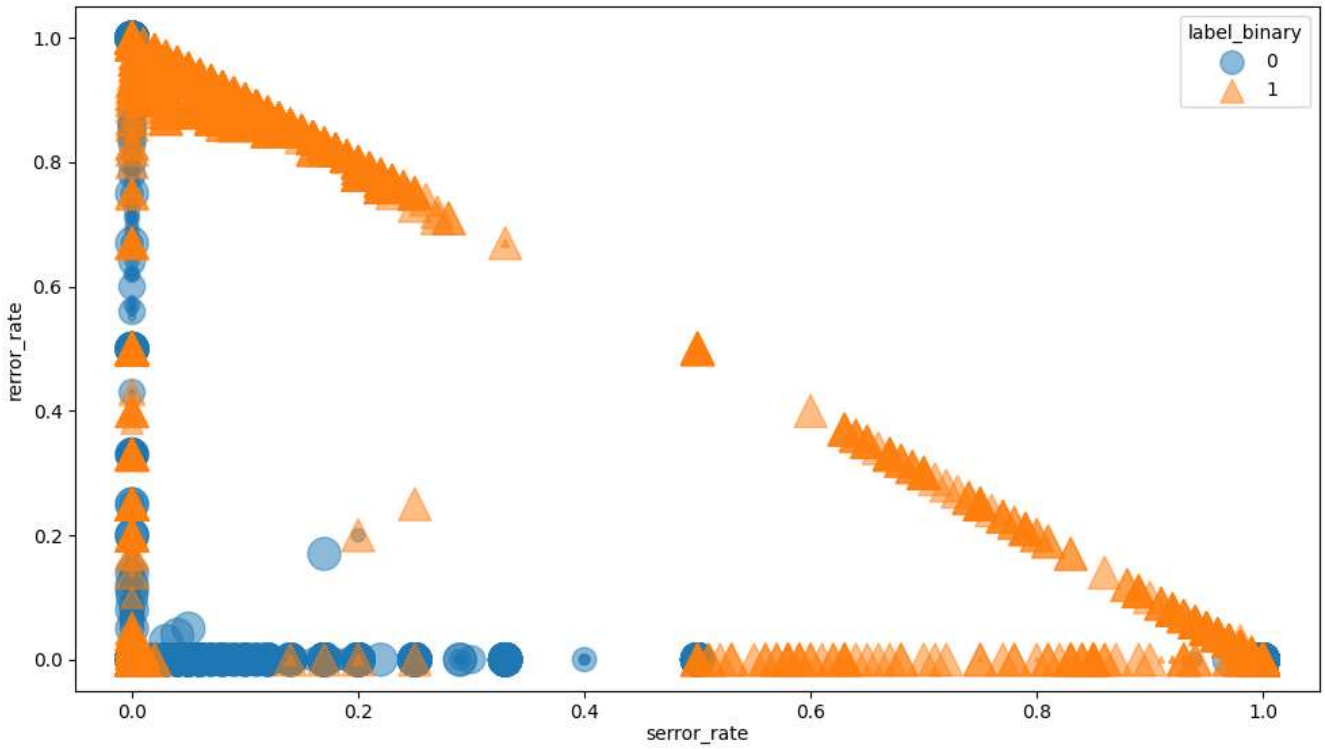

Рис. А.5 Бульбашкова діаграма: rerror_rate та error_rate

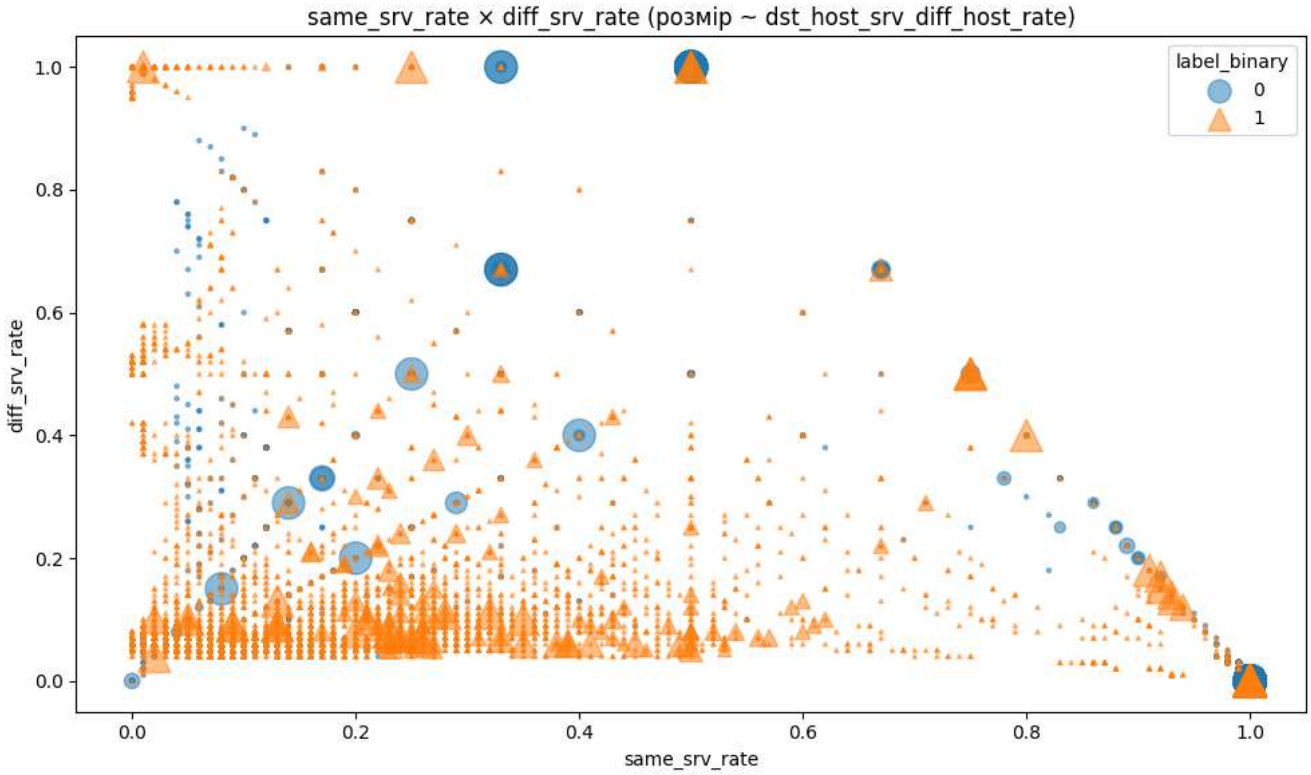

Рис. А.5 Бульбашкова діаграма: same_srv_rate та diff_srv_rate

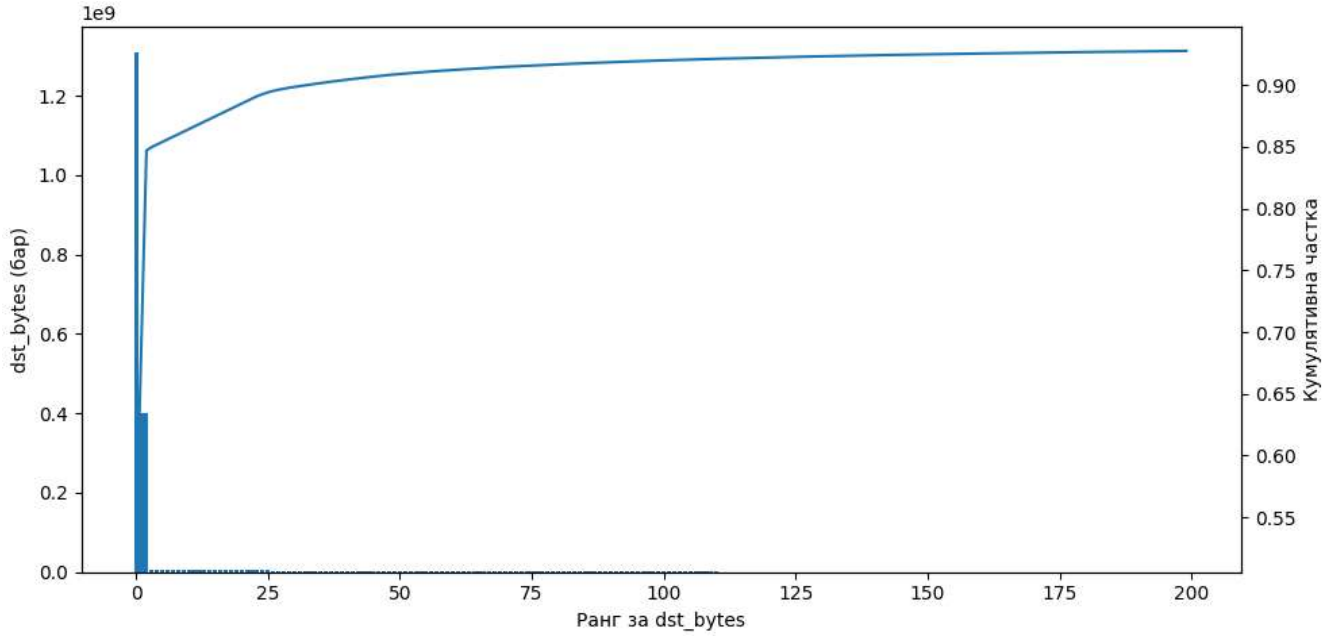

Рис. А.6 Парето-аналіз dst_bytes

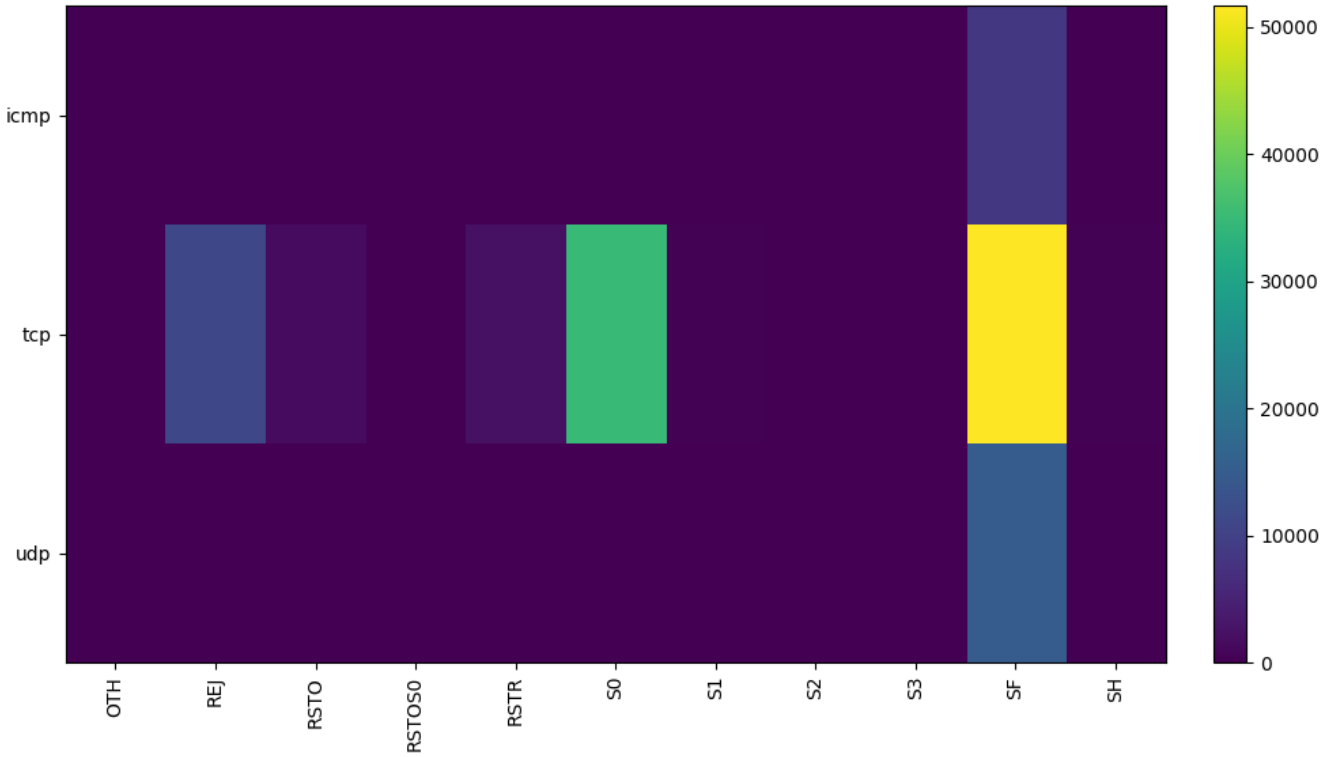

Рис. А.7 Частоти протоколів за типом прапорця: protocol_type × flag

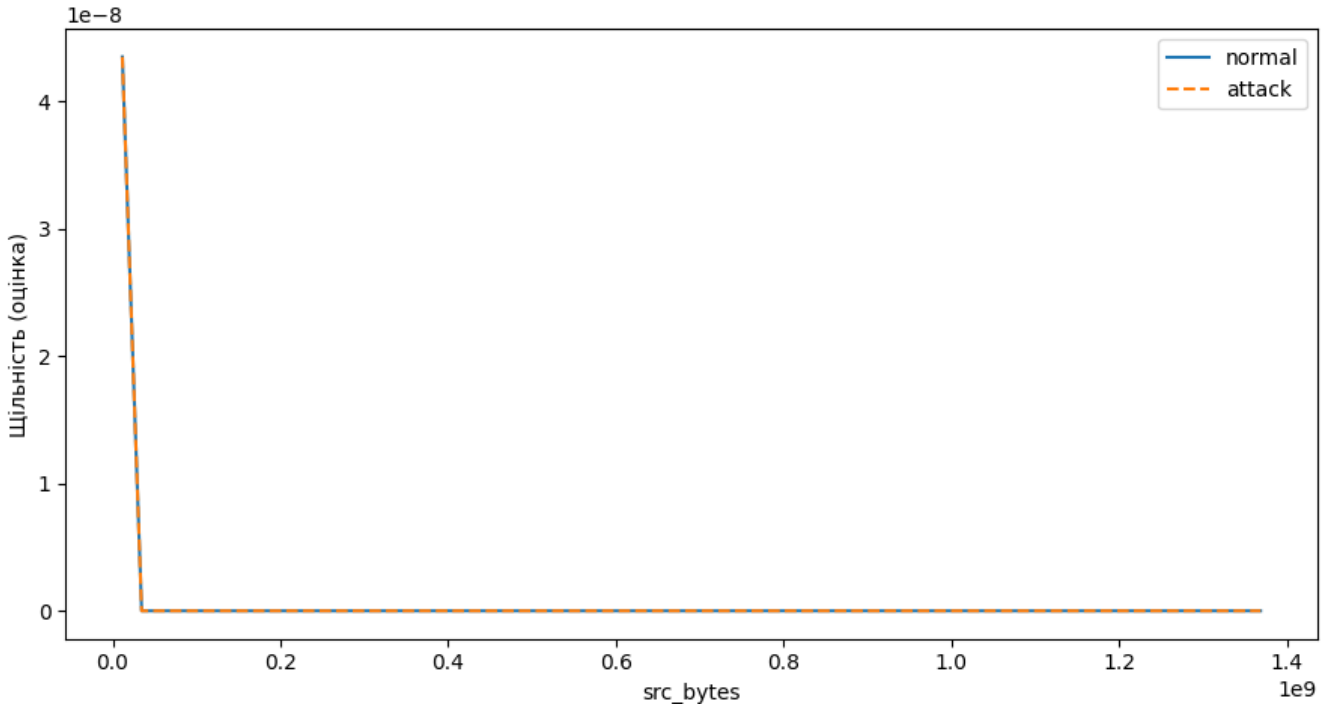

Рис. А.8 Диференціальні гістограми для src_bytes

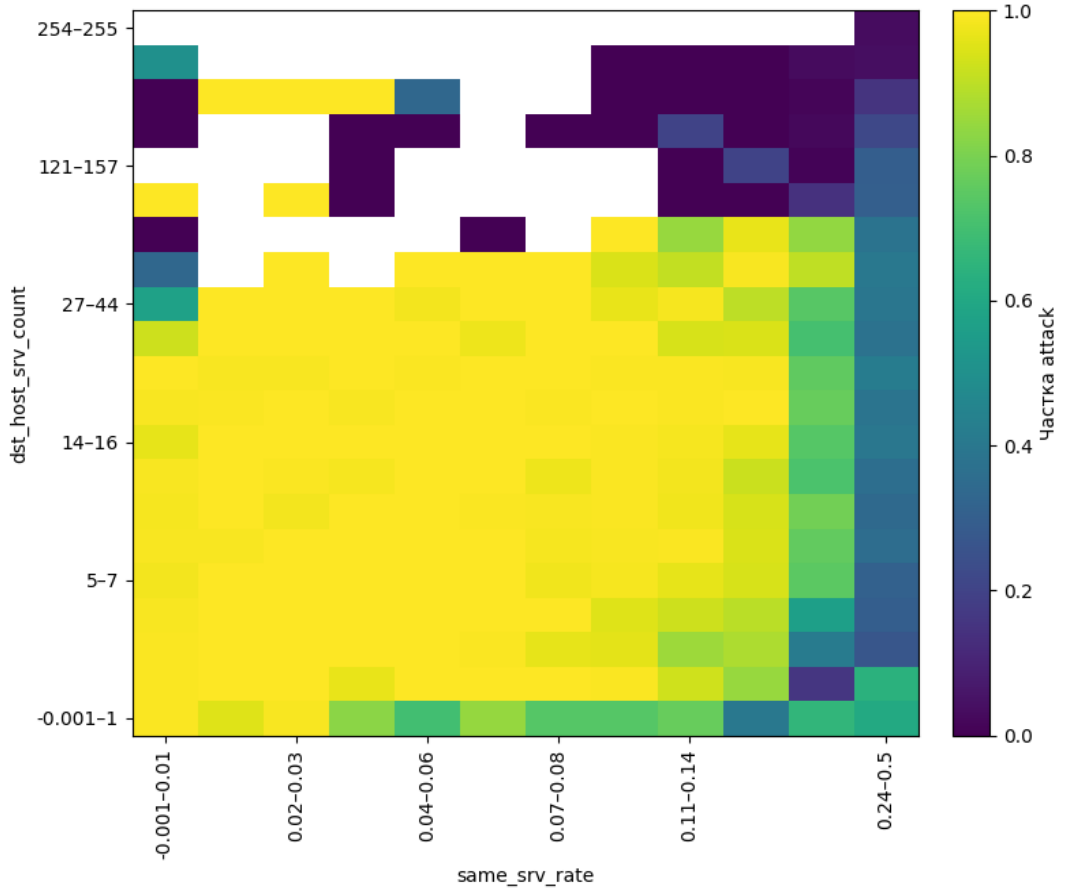

Рис. А.9 2D частка атак

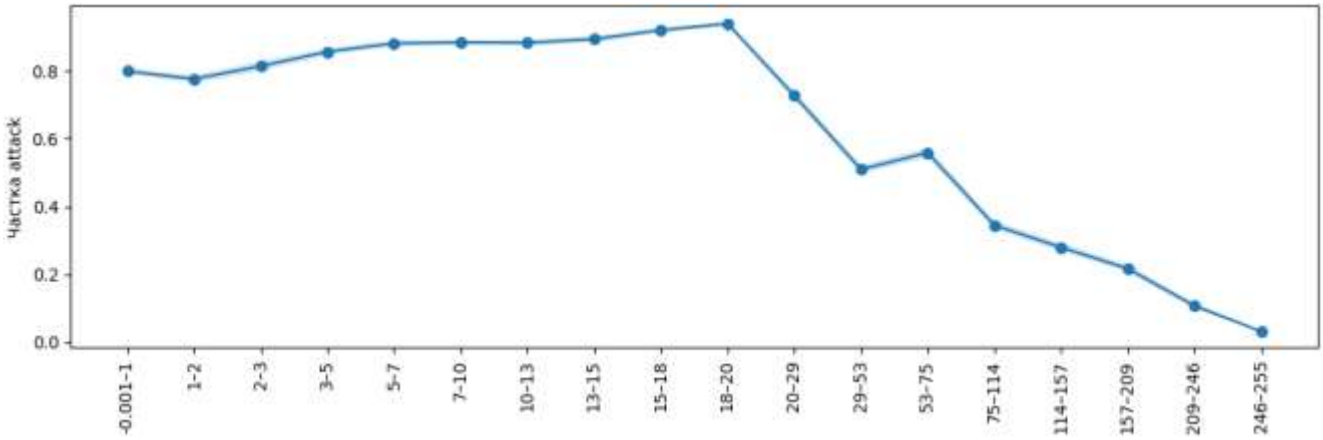

Рис. А.10 Диференціальні гістограми для атрибуту dst_host_src_count

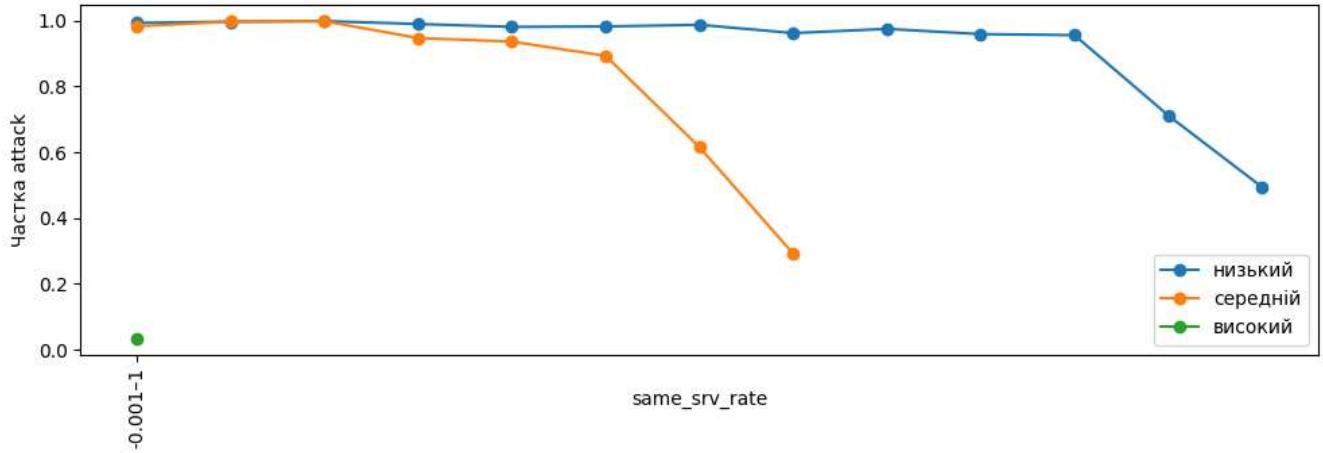

Рис. А.11 Інтеракційні профілі: частка атак від same_srv_rate

Додаток Б. Лістинги програмного коду

Лістинг 1. Код класу «ModelsForm»

```
using Microsoft.ML;
using Microsoft.ML.Data;
using NetworkMonitoringApp.AppCode;
using NetworkMonitoringApp.Forms.Systems;
using NetworkMonitoringApp.Providers;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace NetworkMonitoringApp.Forms.SysMS {
  public partial class ModelsForm : Form {
    private MLContext mlContext;
    ITransformer trainedModel;
    private IDataView dataView;
    private string _Path = "";

    private int _selectedRowIndex = 0;
    private ValidationMy _Validation = new ValidationMy();
    private ModelsProvider _ModelsProvider = new ModelsProvider();
    private List<Models> _ModelsList = new List<Models>();
    private LogsProvider _LogsProvider = new LogsProvider();
    private bool _IsModelTrain = false;
    private Random _rand = new Random();

    public ModelsForm() {
      InitializeComponent();
      DataLoad();
    }

    // Допоміжний метод безпечного дописування тексту у RaportTBox з автоскролом
    private void AppendRaport(string text) {
      if (RaportTBox.InvokeRequired) {
        RaportTBox.Invoke(new Action<string>(AppendRaport), text);
        return;
      }
      RaportTBox.AppendText(text);
      RaportTBox.SelectionStart = RaportTBox.Text.Length;
      RaportTBox.ScrollToCaret();
    }
```

```
// АСИНХРОННИЙ обробник події натискання кнопки "Відкрити"
private async void OpenBtn_Click(object sender, EventArgs e) {
 using (var openFileDialog = new OpenFileDialog {
  Filter = "Text files (*.csv)|*.csv|All files (*.*)|*.*",
  FilterIndex = 2,
  RestoreDirectory = true,
  Title = "Оберіть CSV з мережевим трафіком"
 }) {
  if (openFileDialog.ShowDialog() != DialogResult.OK) return;
  _Path = openFileDialog.FileName;
  FileNameTBox.Text = openFileDialog.FileName;
 }

 mlContext = new MLContext(seed: 0);

 RaportTBox.Clear();
 AppendRaport("Завантаження даних…\r\n");

 try {
  // 1) Завантаження даних
  dataView = await Task.Run(() =>
     mlContext.Data.LoadFromTextFile<NetworkTrafficData>(
        path: _Path, hasHeader: true, separatorChar: ','));

  AppendRaport("Дані завантажено. Формую розподіл класів…\r\n");

  var classDistribution = await Task.Run(() =>
     mlContext.Data
           .CreateEnumerable<NetworkTrafficData>(dataView, reuseRowObject: false)
           .GroupBy(d => d.labels)
           .Select(g => new { Label = g.Key, Count = g.Count() })
           .OrderByDescending(x => x.Count)
           .ToList());

  foreach (var item in classDistribution) {
   AppendRaport($"Клас {item.Label}: {item.Count} прикладів\r\n");
   await Task.Yield();
  }

  AppendRaport("Побудова конвеєра підготовки ознак…\r\n");

  // 2) Пайплайн (копія текстової мітки → кодування ключем → ознаки)
  var dataProcessPipeline = await Task.Run(() =>
     mlContext.Transforms.CopyColumns("LabelText", "Label")
        .Append(mlContext.Transforms.Conversion.MapValueToKey("Label",
"LabelText"))
        .Append(mlContext.Transforms.Categorical.OneHotEncoding(new[]
        {
           new InputOutputColumnPair("protocol_type_encoded", "protocol_type"),
           new InputOutputColumnPair("service_encoded", "service"),
           new InputOutputColumnPair("flag_encoded", "flag"),
           new InputOutputColumnPair("land_encoded", "land"),
```

```
                    new InputOutputColumnPair("logged_in_encoded", "logged_in"),
                    new InputOutputColumnPair("is_host_login_encoded", "is_host_login"),
                    new InputOutputColumnPair("is_guest_login_encoded", "is_guest_login")
                })
                .Append(mlContext.Transforms.Concatenate("Features",
                    "duration", "protocol_type_encoded", "service_encoded",
                    "flag_encoded", "src_bytes", "dst_bytes",
                    "land_encoded", "wrong_fragment", "urgent", "hot",
                    "num_failed_logins", "logged_in_encoded", "num_compromised",
                    "root_shell", "su_attempted", "num_root",
                    "num_file_creations", "num_shells", "num_access_files",
                    "num_outbound_cmds", "is_host_login_encoded", "is_guest_login_encoded",
                    "count", "srv_count", "serror_rate", "srv_serror_rate",
                    "rerror_rate", "srv_rerror_rate", "same_srv_rate",
                    "diff_srv_rate", "srv_diff_host_rate", "dst_host_count",
                    "dst_host_srv_count", "dst_host_same_srv_rate",
                    "dst_host_diff_srv_rate", "dst_host_same_src_port_rate",
"dst_host_srv_diff_host_rate",
                    "dst_host_serror_rate", "dst_host_srv_serror_rate", "dst_host_rerror_rate",
                    "dst_host_srv_rerror_rate"
                ))
                .Append(mlContext.Transforms.NormalizeMinMax("Features"))
                .AppendCacheCheckpoint(mlContext)
            );

            AppendRaport("Розділення на train/test…\r\n");
            var splitData = await Task.Run(() => mlContext.Data.TrainTestSplit(dataView,
testFraction: 0.2));

            AppendRaport("Тренування моделі …\r\n");
            trainedModel = await Task.Run(() => {
              var trainer = mlContext.MulticlassClassification.Trainers
                .LbfgsMaximumEntropy(labelColumnName: "Label", featureColumnName:
"Features")
                .Append(mlContext.Transforms.Conversion.MapKeyToValue("PredictedLabel"));

              var trainingPipeline = dataProcessPipeline.Append(trainer);
              return trainingPipeline.Fit(splitData.TrainSet);
            });

            AppendRaport("Оцінка моделі на тестовому наборі…\r\n");

            // 3) Прогнози та метрики
            var predicted = trainedModel.Transform(splitData.TestSet);

            var metrics = await Task.Run(() =>
               mlContext.MulticlassClassification.Evaluate(predicted, labelColumnName: "Label"));

            var cm = metrics.ConfusionMatrix;
            int k = cm.Counts.Count;

            // Імена класів з даних (LabelKey ↔ LabelText). Якщо чогось бракує — Class_i
```

```
            var classNames = BuildClassNamesFromPredicted(mlContext, predicted, k);

            // ---- ДРУК У ЗВІТ (метрики), БЕЗ друку матриці плутанини ----
            AppendRaport("Метрики мультикласової класифікації:\r\n");
            AppendRaport($"  MicroAccuracy : {metrics.MicroAccuracy:P2}\r\n");
            AppendRaport($"  MacroAccuracy : {metrics.MacroAccuracy:P2}\r\n");
            AppendRaport($"  LogLoss       : {metrics.LogLoss:F4}\r\n");
            if (metrics.PerClassLogLoss != null && metrics.PerClassLogLoss.Count > 0)
              AppendRaport($"  PerClassLogLoss: [{string.Join(", ",
metrics.PerClassLogLoss.Select(v => v.ToString("F4")))}]\r\n");

            var prf = ComputePerClassPrf1(cm);
            var macro = (Precision: prf.Average(x => x.Precision),
                    Recall: prf.Average(x => x.Recall),
                    F1: prf.Average(x => x.F1));
            var micro = ComputeMicroPrf1(cm);

            AppendRaport($"\r\nMacro-avg:  P={macro.Precision:F4}, R={macro.Recall:F4},
F1={macro.F1:F4}\r\n");
            AppendRaport($"Micro-avg:  P={micro.Precision:F4}, R={micro.Recall:F4},
F1={micro.F1:F4}\r\n");

            // 4) Експорти (БЕЗ AUC у файл метрик!)
            //var support = cm.Counts.Select(row => row.Sum()).ToArray(); // <-- це int[] і саме
його передаємо!

            //var outDir =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Desktop), "ml_reports");
            //Directory.CreateDirectory(outDir);

            //ExportConfusionMatrixCsv(Path.Combine(outDir, "confusion_matrix.csv"), cm,
classNames);
            //ExportPerClassMetricsCsv(Path.Combine(outDir, "per_class_metrics.csv"), classNames,
prf, support);

            //AppendRaport($"\r\nЕкспорти збережено у: {outDir}\r\n" +
            //          $" - confusion_matrix.csv\r\n" +
            //          $" - per_class_metrics.csv\r\n");

            _IsModelTrain = true;
            AppendRaport("Готово. Модель навчено та оцінено.\r\n");
          } catch (Exception ex) {
            AppendRaport($"Помилка: {ex.Message}\r\n");
          }
        }

        private void ModelsGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
          if (e.ColumnIndex == 5 && ModelsGridView[0, e.RowIndex].Value.ToString() !=
_ModelsList[0].Message) {
            if (MessageBox.Show("Ви дійсно хочете видалити цю модель?", "Видалити",
MessageBoxButtons.YesNo) == DialogResult.Yes) {
              _ModelsProvider.DeleteModelsByModelsId(Convert.ToInt32(ModelsGridView[0,
```

```
e.RowIndex].Value.ToString()));
                DataLoad();
              }
            }
          }

          // Для зчитування прогнозів із колонок PredictedLabel/Score/LabelText
          private sealed class PredictionWithScores {
            [ColumnName("PredictedLabel")] public uint PredictedLabelKey { get; set; } // 1..K
            [ColumnName("Score")] public float[] Score { get; set; }
            [ColumnName("Label")] public uint LabelKey { get; set; }  // 1..K
            [ColumnName("LabelText")] public string LabelText { get; set; }
          }

          private sealed class LabelKeyAndText {
            [ColumnName("Label")] public uint LabelKey { get; set; }   // 1..K
            [ColumnName("LabelText")] public string LabelText { get; set; }
          }


          private string[] BuildClassNamesFromPredicted(MLContext ml, IDataView predicted, int k)
{
            var map = new Dictionary<uint, string>();
            foreach (var row in ml.Data.CreateEnumerable<LabelKeyAndText>(predicted,
reuseRowObject: false)) {
              if (row.LabelKey == 0) continue;
              if (!map.ContainsKey(row.LabelKey) && !string.IsNullOrWhiteSpace(row.LabelText))
                map[row.LabelKey] = row.LabelText;
            }
            var names = new string[k];
            for (uint key = 1; key <= k; key++)
              names[key - 1] = map.TryGetValue(key, out var name) &&
!string.IsNullOrWhiteSpace(name)
                               ? name : $"Class_{key}";
            return names;
          }

          // 2) Пер-класні та зведені метрики (вже узгоджено з вашими викликами)
          private (double Precision, double Recall, double F1)[]
ComputePerClassPrf1(ConfusionMatrix cm) {
            int k = cm.Counts.Count;
            var res = new (double Precision, double Recall, double F1)[k];
            var counts = cm.Counts;

            for (int c = 0; c < k; c++) {
              double tp = counts[c][c];
              double fn = 0; for (int j = 0; j < k; j++) if (j != c) fn += counts[c][j];
              double fp = 0; for (int i = 0; i < k; i++) if (i != c) fp += counts[i][c];
              double precision = (tp + fp) > 0 ? tp / (tp + fp) : 0.0;
              double recall = (tp + fn) > 0 ? tp / (tp + fn) : 0.0;
              double f1 = (precision + recall) > 0 ? 2 * precision * recall / (precision + recall) : 0.0;
              res[c] = (precision, recall, f1);
```

```
  }
  return res;
 }

 private (double Precision, double Recall, double F1) ComputeMicroPrf1(ConfusionMatrix
cm) {
  int k = cm.Counts.Count;
  var counts = cm.Counts;

  double tpSum = 0, fpSum = 0, fnSum = 0;
  for (int c = 0; c < k; c++) {
   double tp = counts[c][c];
   double fn = 0; for (int j = 0; j < k; j++) if (j != c) fn += counts[c][j];
   double fp = 0; for (int i = 0; i < k; i++) if (i != c) fp += counts[i][c];
   tpSum += tp; fpSum += fp; fnSum += fn;
  }
  double precision = (tpSum + fpSum) > 0 ? tpSum / (tpSum + fpSum) : 0.0;
  double recall = (tpSum + fnSum) > 0 ? tpSum / (tpSum + fnSum) : 0.0;
  double f1 = (precision + recall) > 0 ? 2 * precision * recall / (precision + recall) : 0.0;
  return (precision, recall, f1);
 }

 // 3) Експорт у CSV
 private void ExportConfusionMatrixCsv(string path, ConfusionMatrix cm, string[] labels) {
  var sb = new StringBuilder();
  sb.Append("Actual/Predicted");
  foreach (var lab in labels)
   sb.Append(',').Append("").Append(lab.Replace("\"", "\"\"")).Append("");
  sb.AppendLine();

  for (int i = 0; i < labels.Length; i++) {
   sb.Append("").Append(labels[i].Replace("\"", "\"\"")).Append("");
   foreach (var v in cm.Counts[i])
    sb.Append(',').Append(v);
   sb.AppendLine();
  }
  File.WriteAllText(path, sb.ToString(), new UTF8Encoding(true));
 }

 private void ExportPerClassMetricsCsv(
    string path, string[] labels, (double P, double R, double F1)[] prf, double[] support) {
  var sb = new StringBuilder();
  sb.AppendLine("Label,Support,Precision,Recall,F1");
  for (int i = 0; i < labels.Length; i++) {
   sb.Append("").Append(labels[i].Replace("\"", "\"\"")).Append("").Append(',')
     .Append(((int)Math.Round(support[i]))).Append(',')
     .Append(prf[i].P.ToString("F6", CultureInfo.InvariantCulture)).Append(',')
     .Append(prf[i].R.ToString("F6", CultureInfo.InvariantCulture)).Append(',')
     .Append(prf[i].F1.ToString("F6", CultureInfo.InvariantCulture)).AppendLine();
  }
  File.WriteAllText(path, sb.ToString(), new UTF8Encoding(true));
 }
```

```
        private void SaveBtn_Click(object sender, EventArgs e) {
          if (IsDataEnteringCorrect()) {
            //Зберігання моделі
            string pathName = @"\teach\" + GenerateFileName() + ".zip";
            string localProj =

System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location);
            _ModelsProvider.InsertModels(ModelsNamesTBox.Text, pathName);
            mlContext.Model.Save(trainedModel, dataView.Schema, localProj + pathName);
            ClearAllData();
            _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
              "Було навчено модель " +
              ModelsNamesTBox.Text, DateTime.Now);
            MessageBox.Show("Дані успішно збережено!");
          }
        }

        private void ClearBtn_Click(object sender, EventArgs e) {
          ClearAllData();
        }

        private void ExitBtn_Click(object sender, EventArgs e) {
          this.Close();
        }


        public string GenerateFileName() {
          DateTime now = DateTime.Now;
          string fileName = string.Format("{0}_{1}_{2}_{3}_{4}_{5}",
              now.Year, now.Month, now.Day, now.Hour, now.Minute, now.Second);

          return fileName;
        }

        private void ClearAllData() {
          _IsModelTrain = false;
          ModelsNamesTBox.Text = String.Empty;
          RaportTBox.Text = String.Empty;
          DataLoad();
        }
        private bool IsDataEnteringCorrect() {
          bool isCorrect = true;
          if (!_IsModelTrain) {
            MessageBox.Show("Неможливо зберегти дані. \r\nЩе не навчено модель!",
"Увага!");
            isCorrect = false;
          }
          if (_Validation.IsDataEntering(ModelsNamesTBox.Text)) {
            ModelsNamesValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
          } else {
            ModelsNamesValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
            isCorrect = false;
```

```
      }
      return isCorrect;
    }


    private void DataLoad() {
      int firstRowIndex = 0;
      if (ModelsGridView.FirstDisplayedScrollingRowIndex > 0) {
        firstRowIndex = ModelsGridView.FirstDisplayedScrollingRowIndex;
      }
      try {
        _ModelsList = _ModelsProvider.GetAllModels();
        LoadDataInModelsGridView(_ModelsList);
        if (_selectedRowIndex == ModelsGridView.Rows.Count) {
          _selectedRowIndex = ModelsGridView.Rows.Count - 1;
        }
        if (_selectedRowIndex >= 0) {
          ModelsGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
          ModelsGridView.Rows[_selectedRowIndex].Selected = true;
        }
      } catch (Exception ex) {
        MessageBox.Show(ex.ToString());
      }
    }

    }
    }


  }
}


// Клас для завантаження даних
public class NetworkTrafficData {
  [LoadColumn(0)]
  public float duration;

  [LoadColumn(1)]
  public string protocol_type;

  [LoadColumn(2)]
  public string service;

  [LoadColumn(3)]
  public string flag;

  [LoadColumn(4)]
  public float src_bytes;

  [LoadColumn(5)]
  public float dst_bytes;
```

```
[LoadColumn(6)]
public float land;

[LoadColumn(7)]
public float wrong_fragment;

[LoadColumn(8)]
public float urgent;

[LoadColumn(9)]
public float hot;

[LoadColumn(10)]
public float num_failed_logins;

[LoadColumn(11)]
public float logged_in;

[LoadColumn(12)]
public float num_compromised;

[LoadColumn(13)]
public float root_shell;

[LoadColumn(14)]
public float su_attempted;

[LoadColumn(15)]
public float num_root;

[LoadColumn(16)]
public float num_file_creations;

[LoadColumn(17)]
public float num_shells;

[LoadColumn(18)]
public float num_access_files;

[LoadColumn(19)]
public float num_outbound_cmds;

[LoadColumn(20)]
public float is_host_login;

[LoadColumn(21)]
public float is_guest_login;

[LoadColumn(22)]
public float count;
```

```
[LoadColumn(23)]
public float srv_count;

[LoadColumn(24)]
public float serror_rate;

[LoadColumn(25)]
public float srv_serror_rate;

[LoadColumn(26)]
public float rerror_rate;

[LoadColumn(27)]
public float srv_rerror_rate;

[LoadColumn(28)]
public float same_srv_rate;

[LoadColumn(29)]
public float diff_srv_rate;

[LoadColumn(30)]
public float srv_diff_host_rate;

[LoadColumn(31)]
public float dst_host_count;

[LoadColumn(32)]
public float dst_host_srv_count;

[LoadColumn(33)]
public float dst_host_same_srv_rate;

[LoadColumn(34)]
public float dst_host_diff_srv_rate;

[LoadColumn(35)]
public float dst_host_same_src_port_rate;

[LoadColumn(36)]
public float dst_host_srv_diff_host_rate;

[LoadColumn(37)]
public float dst_host_serror_rate;

[LoadColumn(38)]
public float dst_host_srv_serror_rate;

[LoadColumn(39)]
public float dst_host_rerror_rate;

[LoadColumn(40)]
```

```
  public float dst_host_srv_rerror_rate;

  [LoadColumn(41)]
  [ColumnName("Label")]
  public string labels;
 }

 // Клас для прогнозування
 public class NetworkTrafficPrediction {
  [ColumnName("PredictedLabel")]
  public string PredictedLabels;

  public float[] Score;
 }
```

Лістинг 2. Код класу «MonitoringIncidentForm»

```
 using CsvHelper;
 using CsvHelper.Configuration;
 using NetworkMonitoringApp.AppCode;
 using NetworkMonitoringApp.Forms.Systems;
 using NetworkMonitoringApp.Providers;
 using Microsoft.ML;
 using System;
 using System.Collections.Generic;
 using System.ComponentModel;
 using System.Data;
 using System.Drawing;
 using System.Globalization;
 using System.IO;
 using System.Linq;
 using System.Net.Sockets;
 using System.Security.Cryptography;
 using System.Text;
 using System.Threading.Tasks;
 using System.Windows.Forms;
 using static System.Windows.Forms.VisualStyles.VisualStyleElement.ProgressBar;

 namespace NetworkMonitoringApp.Forms.Controls {
  public partial class MonitoringIncidentForm : Form {
   private ValidationMy _Validation = new ValidationMy();
   private Models _SelectedModels = new Models();
   private MLContext context = new MLContext();
   private PredictionEngine<NetworkTrafficData,
    NetworkTrafficPrediction> predictionEngine;

   private ModelsProvider _ModelsProvider = new ModelsProvider();
   private List<Models> _ModelsList = new List<Models>();
   private bool _IsModelsLoad = false;
   private LogsProvider _LogsProvider = new LogsProvider();
   private string filePath = "data.csv";
   List<NetworkTrafficData> _NetworkTrafficDataList = new List<NetworkTrafficData>();

   public MonitoringIncidentForm() {
```

```
  InitializeComponent();
  LoadAllData();
}

private void LoadAllData() {
  _ModelsList = _ModelsProvider.GetAllModels();
  ModelsCBox.DataSource = _ModelsList;
  ModelsCBox.ValueMember = "ModelsId";
  ModelsCBox.DisplayMember = "ModelsName";
  _IsModelsLoad = true;
  _NetworkTrafficDataList = ReadDataFromCsv(filePath);
  ModelsCBox_SelectedValueChanged(ModelsCBox, EventArgs.Empty);
}


private void ModelsCBox_SelectedValueChanged(object sender, EventArgs e) {
  if (_IsModelsLoad && Convert.ToInt32(ModelsCBox.SelectedValue) > 0) {
    _SelectedModels = _ModelsProvider.SelectedModelsByModelsId(
       Convert.ToInt32(ModelsCBox.SelectedValue));
    LoadData(_SelectedModels.ModelsFileModel);
  }
}

private void LoadData(string FilePath) {
  string localProj = Application.StartupPath + FilePath;
  // Визнаення DataViewSchema для конвеєра підготовки даних і навченої моделі
  DataViewSchema modelSchema;
  // Завантаження моделі
  ITransformer model = context.Model.Load(localProj, out modelSchema);
  //Створення механізму прогнозування
  predictionEngine =
    context.Model.CreatePredictionEngine<NetworkTrafficData,
    NetworkTrafficPrediction>(model);
}

public class NetworkTrafficDataMap : ClassMap<NetworkTrafficData> {
  public NetworkTrafficDataMap() {
    Map(m => m.duration).Index(0);
    Map(m => m.protocol_type).Index(1);
    Map(m => m.service).Index(2);
    Map(m => m.flag).Index(3);
    Map(m => m.src_bytes).Index(4);
    Map(m => m.dst_bytes).Index(5);
    Map(m => m.land).Index(6);
    Map(m => m.wrong_fragment).Index(7);
    Map(m => m.urgent).Index(8);
    Map(m => m.hot).Index(9);
    Map(m => m.num_failed_logins).Index(10);
    Map(m => m.logged_in).Index(11);
    Map(m => m.num_compromised).Index(12);
    Map(m => m.root_shell).Index(13);
    Map(m => m.su_attempted).Index(14);
```

```
    Map(m => m.num_root).Index(15);
    Map(m => m.num_file_creations).Index(16);
    Map(m => m.num_shells).Index(17);
    Map(m => m.num_access_files).Index(18);
    Map(m => m.num_outbound_cmds).Index(19);
    Map(m => m.is_host_login).Index(20);
    Map(m => m.is_guest_login).Index(21);
    Map(m => m.count).Index(22);
    Map(m => m.srv_count).Index(23);
    Map(m => m.serror_rate).Index(24);
    Map(m => m.srv_serror_rate).Index(25);
    Map(m => m.rerror_rate).Index(26);
    Map(m => m.srv_rerror_rate).Index(27);
    Map(m => m.same_srv_rate).Index(28);
    Map(m => m.diff_srv_rate).Index(29);
    Map(m => m.srv_diff_host_rate).Index(30);
    Map(m => m.dst_host_count).Index(31);
    Map(m => m.dst_host_srv_count).Index(32);
    Map(m => m.dst_host_same_srv_rate).Index(33);
    Map(m => m.dst_host_diff_srv_rate).Index(34);
    Map(m => m.dst_host_same_src_port_rate).Index(35);
    Map(m => m.dst_host_srv_diff_host_rate).Index(36);
    Map(m => m.dst_host_serror_rate).Index(37);
    Map(m => m.dst_host_srv_serror_rate).Index(38);
    Map(m => m.dst_host_rerror_rate).Index(39);
    Map(m => m.dst_host_srv_rerror_rate).Index(40);
    Map(m => m.labels).Index(41).Name("labels");
  }
}


public List<NetworkTrafficData> ReadDataFromCsv(string filePath) {
  // Ініціалізуємо список для зберігання результатів
  List<NetworkTrafficData> records;

  // Використовуємо блок з використанням для автоматичного закриття файлу
  using (var reader = new StreamReader(filePath))
  using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture)) {
    // Реєструємо мапу для класу
    csv.Context.RegisterClassMap<NetworkTrafficDataMap>();

    // Зчитуємо всі записи в список
    records = csv.GetRecords<NetworkTrafficData>().ToList();
  }

  return records;
}


private void PredictBtn_Click(object sender, EventArgs e) {
  if (IsAllNetworkTrafficDataCorrect()) {
    NetworkTrafficData testNetworkTrafficData = new NetworkTrafficData {
```

```
                duration = (float)Convert.ToDouble(DurationTBox.Text),
                protocol_type = ProtocolTypeTBox.Text,
                service = ServiceTBox.Text,
                flag = FlagTBox.Text,
                src_bytes = (float)Convert.ToDouble(SrcBytesTBox.Text),
                dst_bytes = (float)Convert.ToDouble(DstBytesTBox.Text),
                land = (float)Convert.ToDouble(LandChBox.Checked),
                wrong_fragment = (float)Convert.ToDouble(WrongFragmentTBox.Text),
                urgent = (float)Convert.ToDouble(UrgentTBox.Text),
                hot = (float)Convert.ToDouble(HotTBox.Text),
                num_failed_logins = (float)Convert.ToDouble(NumFailedLoginsTBox.Text),
                logged_in = (float)Convert.ToDouble(LoggedInChBox.Checked),
                num_compromised = (float)Convert.ToDouble(NumCompromisedTBox.Text),
                root_shell = (float)Convert.ToDouble(RootShellChBox.Checked),
                su_attempted = (float)Convert.ToDouble(SuAttemptedTBox.Text),
                num_root = (float)Convert.ToDouble(NumRootTBox.Text),
                num_file_creations = (float)Convert.ToDouble(NumFileCreationsTBox.Text),
                num_shells = (float)Convert.ToDouble(NumShellsTBox.Text),
                num_access_files = (float)Convert.ToDouble(NumAccessFilesTBox.Text),
                num_outbound_cmds = (float)Convert.ToDouble(NumOutboundCmdsTBox.Text),
                is_host_login = (float)Convert.ToDouble(IsHostLoginChBox.Checked),
                is_guest_login = (float)Convert.ToDouble(IsGuestLoginChBox.Checked),
                count = (float)Convert.ToDouble(CountTBox.Text),
                srv_count = (float)Convert.ToDouble(SrvCountTBox.Text),
                serror_rate = (float)Convert.ToDouble(SerrorRateChBox.Checked),
                srv_serror_rate = (float)Convert.ToDouble(SrvSerrorRateChBox.Checked),
                rerror_rate = (float)Convert.ToDouble(RerrorRateChBox.Checked),
                srv_rerror_rate = (float)Convert.ToDouble(SrvRerrorRateChBox.Checked),
                same_srv_rate = (float)Convert.ToDouble(SameSrvRateChBox.Checked),
                diff_srv_rate = (float)Convert.ToDouble(DiffSrvRateChBox.Checked),
                srv_diff_host_rate = (float)Convert.ToDouble(SrvDiffHostRateChBox.Checked),
                dst_host_count = (float)Convert.ToDouble(DstHostCountTBox.Text),
                dst_host_srv_count = (float)Convert.ToDouble(DstHostSrvCountTBox.Text),
                dst_host_same_srv_rate =
(float)Convert.ToDouble(DstHostSameSrvRateChBox.Checked),
                dst_host_diff_srv_rate = (float)Convert.ToDouble(DstHostDiffSrvRateChBox.Checked),
                dst_host_same_src_port_rate =
(float)Convert.ToDouble(DstHostSameSrcPortRateChBox.Checked),
                dst_host_srv_diff_host_rate =
(float)Convert.ToDouble(DstHostSrvDiffHostRateChBox.Checked),
                dst_host_serror_rate = (float)Convert.ToDouble(DstHostSerrorRateChBox.Checked),
                dst_host_srv_serror_rate =
(float)Convert.ToDouble(DstHostSrvSerrorRateChBox.Checked),
                dst_host_rerror_rate = (float)Convert.ToDouble(DstHostRerrorRateChBox.Checked),
                dst_host_srv_rerror_rate =
(float)Convert.ToDouble(DstHostSrvRerrorRateChBox.Checked)
            };
            var prediction = predictionEngine.Predict(testNetworkTrafficData);
            var answer = new StringBuilder();
            answer.AppendLine("\r\n--- Прогнозування ---");
            if (prediction.PredictedLabels != "normal") {
              answer.AppendLine($"\nВиявлено інцидент: {prediction.PredictedLabels}");
```

```
                answer.AppendLine($"Рекомендації:
{GetRecommendations(prediction.PredictedLabels)}");
              } else {
                answer.AppendLine("Не виявлено інциденту.");
                answer.AppendLine($"Рекомендації:
{GetRecommendations(prediction.PredictedLabels)}");
              }
              MonitoringTBox.Text += answer.ToString();
            }
          }

          private void GenBtn_Click(object sender, EventArgs e) {
            if (IsModelCorrect()) {
              if (MoniroringTimer.Enabled) {
                MoniroringTimer.Enabled = false;
                GenBtn.Text = "Моніторити";
                _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
                  "Було зупинено моніторинг моделі " +
                  ModelsCBox.Text, DateTime.Now);
              } else {
                MoniroringTimer.Enabled = true;
                GenBtn.Text = "Зупинити";
                _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
                "Було запущено моніторинг моделі " +
                ModelsCBox.Text, DateTime.Now);
              }
            }
          }


          private void MoniroringTimer_Tick(object sender, EventArgs e) {
            if (_NetworkTrafficDataList.Any()) {
              // Вибір випадкового запису
              Random rand = new Random();
              int index = rand.Next(_NetworkTrafficDataList.Count);
              NetworkTrafficData randomData = _NetworkTrafficDataList[index];
              // Виведення вибраного запису в TextBox
              MonitoringTBox.Invoke((MethodInvoker)(() => {
                var dataInfo = new StringBuilder();
                dataInfo.AppendLine($"Тривалість з'єднання: {randomData.duration} секунд");
                dataInfo.AppendLine($"Тип протоколу: {randomData.protocol_type}");
                dataInfo.AppendLine($"Сервіс: {randomData.service}");
                dataInfo.AppendLine($"Стан з'єднання: {randomData.flag}");
                dataInfo.AppendLine($"Байти від джерела: {randomData.src_bytes}");
                dataInfo.AppendLine($"Байти до призначення: {randomData.dst_bytes}");
                dataInfo.AppendLine($"Індикатор \"land\"-атаки: {randomData.land}");
                dataInfo.AppendLine($"Кількість неправильних фрагментів:
{randomData.wrong_fragment}");
                dataInfo.AppendLine($"Кількість \"urgent\" пакетів: {randomData.urgent}");
                dataInfo.AppendLine($"Кількість \"hot\" показників: {randomData.hot}");
                dataInfo.AppendLine($"Кількість невдалих входів:
{randomData.num_failed_logins}");
```

```
            dataInfo.AppendLine($"Чи виконано вхід: {randomData.logged_in}");
            dataInfo.AppendLine($"Кількість скомпрометованих показників:
{randomData.num_compromised}");
            dataInfo.AppendLine($"Чи активована оболонка root: {randomData.root_shell}");
            dataInfo.AppendLine($"Кількість спроб використання команди su:
{randomData.su_attempted}");
            dataInfo.AppendLine($"Кількість процесів від імені root: {randomData.num_root}");
            dataInfo.AppendLine($"Кількість створених файлів:
{randomData.num_file_creations}");
            dataInfo.AppendLine($"Кількість піднятих оболонок: {randomData.num_shells}");
            dataInfo.AppendLine($"Кількість доступів до файлів:
{randomData.num_access_files}");
            dataInfo.AppendLine($"Кількість вихідних команд в FTP сесіях:
{randomData.num_outbound_cmds}");
            dataInfo.AppendLine($"Чи це хостовий вхід: {randomData.is_host_login}");
            dataInfo.AppendLine($"Чи це гостьовий вхід: {randomData.is_guest_login}");
            dataInfo.AppendLine($"Кількість з'єднань до тієї ж IP-адреси: {randomData.count}");
            dataInfo.AppendLine($"Кількість з'єднань до того ж сервісу:
{randomData.srv_count}");
            dataInfo.AppendLine($"Частота помилок SYN у з'єднаннях:
{randomData.serror_rate}");
            dataInfo.AppendLine($"Частота помилок SYN у з'єднаннях до того ж сервісу:
{randomData.srv_serror_rate}");
            dataInfo.AppendLine($"Частота помилок RST у з'єднаннях:
{randomData.rerror_rate}");
            dataInfo.AppendLine($"Частота помилок RST у з'єднаннях до того ж сервісу:
{randomData.srv_rerror_rate}");
            dataInfo.AppendLine($"Частка з'єднань до того ж сервісу:
{randomData.same_srv_rate}");
            dataInfo.AppendLine($"Частка з'єднань до інших сервісів:
{randomData.diff_srv_rate}");
            dataInfo.AppendLine($"Частка з'єднань до інших хостів, що використовують той же
сервіс: {randomData.srv_diff_host_rate}");
            dataInfo.AppendLine($"Кількість з'єднань до того ж хоста:
{randomData.dst_host_count}");
            dataInfo.AppendLine($"Кількість з'єднань до того ж сервісу на тому ж хості:
{randomData.dst_host_srv_count}");
            dataInfo.AppendLine($"Частка з'єднань до того ж сервісу на тому ж хості:
{randomData.dst_host_same_srv_rate}");
            dataInfo.AppendLine($"Частка з'єднань до інших сервісів на тому ж хості:
{randomData.dst_host_diff_srv_rate}");
            dataInfo.AppendLine($"Частка з'єднань з того ж джерела на тому ж хості через той
же порт: {randomData.dst_host_same_src_port_rate}");
            dataInfo.AppendLine($"Частка з'єднань до інших хостів з того ж сервісу:
{randomData.dst_host_srv_diff_host_rate}");
            dataInfo.AppendLine($"Частота помилок SYN на тому ж хості:
{randomData.dst_host_serror_rate}");
            dataInfo.AppendLine($"Частота помилок SYN на тому ж сервісі і хості:
{randomData.dst_host_srv_serror_rate}");
            dataInfo.AppendLine($"Частота помилок RST на тому ж хості:
{randomData.dst_host_rerror_rate}");
            dataInfo.AppendLine($"Частота помилок RST на тому ж сервісі і хості:
```

```
{randomData.dst_host_srv_rerror_rate}");
                dataInfo.AppendLine($"Мітка: {randomData.labels}");

                MonitoringTBox.Text = dataInfo.ToString();
                // Прогнозування на основі вибраних даних
                var prediction = predictionEngine.Predict(randomData);
                var answer = new StringBuilder();
                answer.AppendLine("\r\n--- Прогнозування ---");
                if (prediction.PredictedLabels != "normal") {
                  answer.AppendLine($"\nВиявлено інцидент: {prediction.PredictedLabels}");
                  answer.AppendLine($"Рекомендації:
{GetRecommendations(prediction.PredictedLabels)}");
                } else {
                  answer.AppendLine("Не виявлено інциденту.");
                  answer.AppendLine($"Рекомендації:
{GetRecommendations(prediction.PredictedLabels)}");
                }
                // Виведення результату прогнозування
                MonitoringTBox.Text += answer.ToString();
              }));
            }
          }


          private bool IsAllNetworkTrafficDataCorrect() {
            bool isCorrect = true;
            if (Convert.ToInt32(ModelsCBox.SelectedValue) > 0) {
              ModelsValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
            } else {
              ModelsValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
              isCorrect = false;
            }

            if (_Validation.IsDataConvertToInt(DurationTBox.Text)) {
              DurationValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
            } else {
              DurationValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
              isCorrect = false;
            }
            if (_Validation.IsDataEntering(ProtocolTypeTBox.Text)) {
              ProtocolTypeValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
            } else {
              ProtocolTypeValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
              isCorrect = false;
            }
            if (_Validation.IsDataEntering(ServiceTBox.Text)) {
              ServiceValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
            } else {
              ServiceValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
              isCorrect = false;
            }
            if (_Validation.IsDataEntering(FlagTBox.Text)) {
```

```
  FlagValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  FlagValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(SrcBytesTBox.Text)) {
  SrcBytesValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  SrcBytesValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(DstBytesTBox.Text)) {
  DstBytesValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  DstBytesValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(WrongFragmentTBox.Text)) {
  WrongFragmentValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  WrongFragmentValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(UrgentTBox.Text)) {
  UrgentValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  UrgentValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(HotTBox.Text)) {
  HotValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  HotValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(NumFailedLoginsTBox.Text)) {
  NumFailedLoginsValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  NumFailedLoginsValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(NumCompromisedTBox.Text)) {
  NumCompromisedValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  NumCompromisedValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
 }
 if (_Validation.IsDataConvertToDouble(SuAttemptedTBox.Text)) {
  SuAttemptedValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
 } else {
  SuAttemptedValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
```

```
            }
            if (_Validation.IsDataConvertToDouble(NumRootTBox.Text)) {
              NumRootValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
            } else {
              NumRootValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
              isCorrect = false;
            }
            if (_Validation.IsDataConvertToDouble(NumFileCreationsTBox.Text)) {
              NumFileCreationsValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
            } else {
              NumFileCreationsValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
              isCorrect = false;
            }
            if (_Validation.IsDataConvertToDouble(NumShellsTBox.Text)) {
              NumShellsValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
            } else {
              NumShellsValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
              isCorrect = false;
            }
            if (_Validation.IsDataConvertToDouble(NumAccessFilesTBox.Text)) {
              NumAccessFilesValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
            } else {
              NumAccessFilesValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
              isCorrect = false;
            }
            if (_Validation.IsDataConvertToDouble(NumOutboundCmdsTBox.Text)) {
              NumOutboundCmdsValidationLbl.Text =
NamesMy.ProgramButtons.RequiredValidation;
            } else {
              NumOutboundCmdsValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
              isCorrect = false;
            }
            if (_Validation.IsDataConvertToDouble(CountTBox.Text)) {
              CountValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
            } else {
              CountValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
              isCorrect = false;
            }
            if (_Validation.IsDataConvertToDouble(SrvCountTBox.Text)) {
              SrvCountValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
            } else {
              SrvCountValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
              isCorrect = false;
            }
            if (_Validation.IsDataConvertToDouble(DstHostCountTBox.Text)) {
              DstHostCountValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
            } else {
              DstHostCountValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
              isCorrect = false;
            }
            if (_Validation.IsDataConvertToDouble(DstHostSrvCountTBox.Text)) {
              DstHostSrvCountValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
```

```
        } else {
          DstHostSrvCountValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
          isCorrect = false;
        }
        return isCorrect;
      }

      private bool IsModelCorrect() {
        bool isCorrect = true;
        if (Convert.ToInt32(ModelsCBox.SelectedValue) > 0) {
          ModelsValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
        } else {
          ModelsValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
          isCorrect = false;
        }
        return isCorrect;
      }



      // Метод для отримання рекомендацій на основі типу інциденту
      private string GetRecommendations(string incidentType) {
        switch (incidentType.ToLower()) {
          case "normal":
            return "Все в порядку. Жодних підозрілих активностей не виявлено.";

          case "portsweep":
            return "Виявлено сканування портів. Рекомендується обмежити доступ до портів
або використовувати захист мережі (Firewall) для блокування подібної активності.";

          case "neptune":
            return "Виявлено можливу DDoS-атаку (SYN flood). Перевірте можливість атаки на
сервіс і заблокуйте підозрілі IP-адреси.";

          case "smurf":
            return "Виявлено можливу Smurf-атаку. Необхідно заборонити передачу ICMP
Echo запитів і захистити маршрутизатори від широкомовних запитів.";

          case "satan":
            return "Система піддається скануванню на вразливості (SATAN-атака).
Рекомендується провести аудит безпеки і зміцнити найвразливіші служби.";

          case "apache2":
            return "Можлива DoS-атака на веб-сервер Apache. Перевірте журнали серверу і
застосуйте патчі для його захисту.";

          case "teardrop":
            return "Виявлено Teardrop-атаку (фрагментація пакетів). Рекомендується оновити
системні патчі та активувати фільтрування на рівні мережі для захисту.";

          case "guess_passwd":
            return "Виявлено спробу підбору паролів. Змініть вразливі паролі і впровадьте
```

двофакторну аутентифікацію.";

case "ipsweep":
return "Виявлено IP-сканування. Необхідно моніторити трафік і заблокувати підозрілі IP-адреси, які виконують сканування.";

case "warezclient":
return "Підозра на незаконне завантаження файлів через Warez-клієнта. Змініть паролі до FTP і заблокуйте доступ.";

case "nmap":
return "Виявлено сканування мережі через Nmap. Рекомендується блокувати підозрілі IP і налаштувати захист на рівні мережевого трафіку.";

case "warezmaster":
return "Загроза використання сервера для нелегального розповсюдження файлів. Проведіть аудит доступу до системи і впровадьте захист.";

case "mscan":
return "Виявлено інструмент Mscan, який використовується для масових сканувань мережі. Заблокуйте IP-адреси, з яких відбувається сканування.";

case "back":
return "Зворотне з'єднання може бути використане для віддаленого управління. Перевірте мережеву активність і обмежте доступ до критичних систем.";

case "pod":
return "Виявлено Ping of Death атаку. Необхідно заблокувати вхідні пакети з ненормально великим розміром та перевірити безпеку мережевих пристроїв.";

case "httptunnel":
return "Виявлено використання HTTP Tunnel для обходу захисту. Необхідно моніторити нестандартний трафік і обмежити доступ.";

case "processtable":
return "Можлива атака, що призводить до заповнення таблиці процесів (DoS). Перевірте навантаження на систему та заблокуйте підозрілий трафік.";

case "mailbomb":
return "Виявлено спробу розсилки великої кількості листів для перевантаження поштової системи. Встановіть обмеження на кількість вхідних повідомлень та блокуйте IP.";

case "snmpguess":
return "Спроба підбору SNMP-паролів. Рекомендується змінити паролі на більш складні та обмежити доступ до SNMP.";

case "saint":
return "Можлива спроба сканування мережі за допомогою SAINT. Проведіть аудит вразливих сервісів і обмежте доступ до відкритих портів.";

case "multihop":
return "Підозра на використання багатоступеневого з'єднання для приховування

слідів. Моніторте вихідний трафік і заблокуйте доступ до зовнішніх ресурсів.";

case "snmpgetattack":
return "Атака на протокол SNMP. Рекомендується змінити паролі та обмежити доступ до управлінських мережевих інтерфейсів.";

case "buffer_overflow":
return "Виявлено спробу переповнення буфера. Застосуйте оновлення безпеки для вразливих програм.";

case "xsnoop":
return "Підозра на прослуховування трафіку (snooping). Захистіть мережу за допомогою шифрування і моніторте підозрілу активність.";

case "imap":
return "Виявлено спробу атаки на IMAP-сервер. Перевірте налаштування доступу до сервера та змініть паролі на безпечніші.";

case "ps":
return "Можливе використання інструмента ps для збору інформації про процеси. Проведіть аудит прав доступу і моніторте активність користувачів.";

case "rootkit":
return "Виявлено ймовірний rootkit. Рекомендується перевірити систему на наявність шкідливого ПЗ і відновити цілісність системних файлів.";

case "land":
return "Виявлено Land-атаку (посилання пакету із співпадаючими IP і портами джерела та призначення). Заблокуйте трафік з підозрілих джерел.";

case "xterm":
return "Виявлено спробу запуску віддаленого терміналу через X11. Перевірте налаштування доступу до системи і закрийте незахищені порти.";

case "sendmail":
return "Виявлено підозрілу активність з використанням Sendmail. Перевірте журнали поштового сервера і застосуйте патчі безпеки.";

case "phf":
return "Спроба експлуатації уразливості в старих версіях CGI-скриптів. Заблокуйте підозрілий трафік та оновіть веб-сервер.";

case "loadmodule":
return "Виявлено спробу використання уразливостей завантажувальних модулів. Оновіть систему і застосуйте патчі.";

case "perl":
return "Підозра на виконання небезпечного Perl-коду. Обмежте можливість виконання сторонніх скриптів та перевірте систему на наявність вразливостей.";

case "xlock":
return "Атака з метою блокування X11-дисплеїв. Захистіть систему шляхом

```
обмеження доступу до X11-сервісів.";

            case "ftp_write":
              return "Можлива спроба несанкціонованого запису через FTP. Змініть права доступу і застосуйте шифрування з'єднань.";

            case "named":
              return "Виявлено підозрілу активність через DNS-сервер (named). Перевірте конфігурацію сервера і заблокуйте підозрілі запити.";

            default:
              return "Перевірте підозрілу активність та прийміть відповідні заходи.";
          }
        }


      }
    }
```