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:
ДОСЛІДЖЕННЯ МЕТОДІВ ТА РОЗРОБКА РЕКОМЕНДАЦІЙ ЩОДО ВИКОРИСТАННЯ ТЕХНОГОЛОГІЇ ШТУЧНОГО ІНТЕЛЕКТУ В ІНФОРМАЦІЙНІЙ БЕЗПЕЦІ

Виконав
студент групи БІКСм-1-24-1.4д

Федорук Гліб Вікторович
(прізвище, ім’я, по батькові)

(підпис)

Науковий керівник
Кандидат військових наук, 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)
« ___ » _____ 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Федоруку Глібу Вікторовичу

(прізвище, ім'я, по батькові)

1. Тема роботи: Дослідження методів та розробка рекомендацій щодо використання технології штучного інтелекту в інформаційній безпеці;
керівник ___Аносов А. О.____, кандидат військових наук, доцент;

затверджені наказом ректора від «__» _____ 2024 року №__.

2. Термін подання студентом роботи «__» _____ 2025 р.
3. Вихідні дані до роботи: аналітичні звіти про кібератаки, наукові публікації з питань застосування штучного інтелекту в інформаційній безпеці та набір даних CyberFedDefender, використаний для навчання й тестування моделей.

4. Зміст текстової частини роботи (перелік питань, які потрібно розробити):
   4.1. Аналіз предметної області.
   4.2. Розробка методичних основ використання штучного інтелекту в інформаційній безпеці.
   4.3. Програмна реалізація та експериментальне дослідження.
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 Федорук Гліб Вікторович
(прізвище, ім'я, по батькові)

Науковий керівник Аносов Андрій Олександрович
(прізвище, ім'я, по батькові)

РЕФЕРАТ

Кваліфікаційна робота присвячена технологіям використання штучного інтелекту та методів машинного навчання в системах виявлення та моніторингу кіберзагроз у мережевому трафіку.

Робота складається зі вступу, трьох розділів, що містять 45 рисунків та 9 таблиць, висновків та списку використаних джерел, що містить 50 найменувань.

Загальний обсяг роботи становить 126 сторінок, з яких 23 сторінки займають ілюстрації і таблиці на окремих аркушах, а також додатки та список використаних джерел.

Об’єктом дослідження є процес виявлення та класифікації кіберзагроз у інформаційних системах засобами штучного інтелекту.

Предметом дослідження є метод застосування алгоритмів машинного навчання для виявлення шкідливої активності у мережевому середовищі.

Метою роботи є підвищення рівня інформаційної безпеки сучасних інформаційних систем шляхом створення інтелектуальної моделі виявлення та прогнозування кіберзагроз у мережевому трафіку.

Наукова новизна одержаних результатів полягає в тому, що в роботі запропоновано методичний підхід до застосування алгоритмів машинного навчання у задачі класифікації мережевого трафіку, який поєднує процедури нормалізації, статистичного аналізу та ансамблевого навчання, та отримано підвищення точності виявлення шкідливої активності до 84,8% завдяки використанню комбінованих ознак і оптимізації параметрів моделей.

Галузь застосування. Запропоновані підходи можуть бути використані для створення інтелектуальних систем кіберзахисту корпоративних мереж, автоматизованих платформ моніторингу трафіку, а також комплексів забезпечення безпеки об’єктів критичної інфраструктури.

Ключові слова: БЕЗПЕКА, ЗАГРОЗА, ІНФОРМАЦІЯ, ІНФОРМАЦІЙНО-АНАЛІТИЧНА СИСТЕМА, ОБ’ЄКТ БЕЗПЕКИ, ПОРУШНИК, СИСТЕМА ЗАХИСТУ.

ЗМІСТ

ВСТУП

Актуальність теми. За даними джерела [1], у 2020 році малий та середній бізнес зазнав понад 700 000 кібератак, що спричинило економічні втрати у розмірі близько 2,8 млрд доларів США. При цьому майже 40 % таких підприємств повідомили про втрату критично важливих даних, що підтверджує вразливість цього сектору до сучасних кіберзагроз.

Згідно з результатами дослідження [2], після пандемії COVID-19 спостерігалося інтенсивне зростання використання дистанційної роботи, що створило нові вектори атак. Серед найбільш поширених загроз виділяються фішинг, несанкціонований віддалений доступ, використання слабких аутентифікаційних механізмів та недостатній рівень захисту кінцевих пристроїв.

У джерелі [3] наголошується, що технології штучного інтелекту дедалі активніше впроваджуються у системи кіберзахисту. Їх застосування охоплює автоматизацію аналізу великих обсягів логів та мережевого трафіку, виявлення аномалій, зменшення кількості хибних спрацювань та підвищення швидкості реагування на інциденти. Водночас зазначається, що значна частина таких рішень не пристосована до середовищ із обмеженими обчислювальними ресурсами чи недостатньою кількістю навчальних даних, що знижує їхню практичну цінність.

Як вказується у звіті [4], попри потенціал AI-технологій для підвищення рівня безпеки, існує низка ризиків, пов’язаних із їх застосуванням. До основних належать непрозорість прийняття рішень, складність інтерпретації результатів, ймовірність упереджень у даних та модельних помилок, а також правові й етичні аспекти, що залишаються неврегульованими.

Виявлені проблеми можна узагальнити у кількох площинах. По-перше, більшість моделей навчаються на репрезентативних, але спрощених наборах даних, тоді як у реальних умовах мережевого трафіку інформація є шумною, неповною або недостатньо маркованою, що знижує точність класифікації. По-друге, багато сучасних методів, особливо глибокі нейронні мережі та ансамблеві алгоритми, характеризуються значними обчислювальними витратами, які є надмірними для

малого бізнесу та організацій з обмеженим бюджетом. По-третє, відсутні універсальні рекомендації щодо вибору оптимальних методів, параметрів моделей, балансування між хибнопозитивними та хибнонегативними прогнозами, а також процедур інтеграції в різні типи середовищ – від корпоративних мереж до хмарних систем та IoT-інфраструктури. Нерозв'язаними залишаються питання регуляції та етики, зокрема забезпечення прозорості й захисту персональних даних у процесі функціонування таких систем.

З урахуванням зазначених викликів застосування технологій штучного інтелекту в інформаційній безпеці є не лише перспективним, а й необхідним напрямом розвитку. Використання інтелектуальних методів дозволяє підвищити рівень захисту інформаційних систем, скоротити час реагування на атаки, знизити кількість хибних спрацювань та забезпечити виявлення загроз у реальному часі. Практична реалізація рекомендацій, сформованих у межах даної роботи, може стати вагомою підтримкою для організацій із обмеженими ресурсами, зокрема малих і середніх підприємств, державних та освітніх установ, допомагаючи їм обрати оптимальні архітектури і методи захисту.

З наукової точки зору відсутність систематизованих порівнянь та методичних рекомендацій щодо застосування AI-технологій у сфері кіберзахисту створює істотну прогалину у знаннях. Дослідження покликане заповнити цю прогалину шляхом формування критеріїв оцінювання, розробки методичних підходів та визначення сценаріїв застосування, що у підсумку сприятиме створенню більш стійких і адаптивних систем інформаційної безпеки.

Мета роботи полягає у розробленні науково обґрунтованих рекомендацій щодо використання технологій штучного інтелекту для підвищення рівня інформаційної безпеки, що забезпечить своєчасне виявлення та нейтралізацію кіберзагроз у сучасних інформаційних системах.

Для досягнення поставленої мети необхідно вирішити наступні **завдання**:

– виконати аналіз сучасних методів і підходів застосування штучного інтелекту в інформаційній безпеці та узагальнити їх сильні й слабкі сторони;

– дослідити основні типи кіберзагроз і вразливостей інформаційних систем, що підлягають автоматизованому виявленню;

– розробити та реалізувати методичні основи застосування моделей машинного навчання для класифікації мережевих загроз із визначенням відповідних критеріїв якості;

– провести експериментальні дослідження роботи розробленого програмного модуля, здійснити оцінку його результатів і сформулювати практичні рекомендації щодо використання технологій штучного інтелекту в системах кіберзахисту.

Виходячи з цього, **об'єктом дослідження** є процес виявлення та класифікації кіберзагроз у інформаційних системах засобами штучного інтелекту. **Предметом дослідження** є метод застосування алгоритмів машинного навчання для виявлення шкідливої активності у мережевому середовищі.

Методи дослідження. Для вирішення поставлених завдань у роботі використано наступні методи: методи машинного навчання для побудови моделей класифікації мережевого трафіку, експериментальне моделювання для перевірки роботи алгоритмів у середовищі симульованих загроз, порівняльний аналіз для оцінки результатів застосування різних моделей, а також методи статистичної обробки даних для визначення метрик якості та узагальнення результатів дослідження.

Наукова новизна одержаних результатів. Наукова новизна одержаних результатів полягає у розробленні методичного підходу до застосування алгоритмів машинного навчання у задачі класифікації мережевого трафіку, який поєднує процедури нормалізації, статистичного аналізу та ансамблевого навчання. Запропоноване рішення забезпечує підвищення точності виявлення шкідливої активності в мережі за рахунок використання комбінованих ознак потоків даних і оптимізації параметрів моделей логістичної регресії, швидкого дерева та градієнтного бустингу. Отримані результати дозволяють знизити кількість хибнопозитивних спрацювань, підвищити достовірність класифікації та

забезпечити стійке функціонування системи кіберзахисту в умовах високої динаміки сучасних кіберзагроз.

Теоретичне та практичне значення роботи полягає у розробці методичних основ застосування технологій штучного інтелекту для виявлення та класифікації кіберзагроз у мережевому трафіку. Теоретичне значення роботи полягає в обґрунтуванні доцільності використання алгоритмів машинного навчання та статистичних методів для підвищення точності і надійності ідентифікації шкідливої активності в умовах змінних характеристик інформаційного середовища. Практичне значення полягає у створенні програмного модуля для автоматизованого аналізу мережевого трафіку, який може бути інтегрований у системи інформаційної безпеки підприємств і організацій з метою своєчасного виявлення атак та підвищення рівня захищеності інформаційних ресурсів.

Галузь застосування. Запропоновані підходи можуть бути використані для створення інтелектуальних систем кіберзахисту корпоративних мереж, автоматизованих платформ моніторингу трафіку, а також комплексів забезпечення безпеки об’єктів критичної інфраструктури.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд сучасних методів і підходів застосування штучного інтелекту в інформаційній безпеці

Штучний інтелект охоплює сукупність методів і алгоритмів, що дозволяють комп'ютерним системам виконувати завдання, які традиційно вимагають участі людини та її інтелектуальних здібностей. До таких завдань належать прийняття рішень у складних ситуаціях, аналіз великих обсягів інформації, прогнозування подій та адаптація до нових умов функціонування. Згідно з визначенням Національного центру кібербезпеки Великої Британії, штучний інтелект – це «комп'ютерні системи, які можуть виконувати завдання, що зазвичай потребують людського інтелекту, зокрема зорове сприйняття, генерацію текстів, розпізнавання мовлення або переклад між мовами» [5]. Така інтерпретація підкреслює багатогранність можливостей ШІ, які охоплюють як когнітивні процеси, так і творчі аспекти інтелектуальної діяльності.

Особливої уваги останнім часом набули генеративні моделі, які здатні створювати новий контент у вигляді текстів, зображень чи відео. Вони широко використовуються для освітніх, наукових та інженерних цілей, проте одночасно становлять загрозу, оскільки можуть бути застосовані зловмисниками для виготовлення дезінформації, маніпулятивних матеріалів або фішингових атак. У цьому контексті ШІ виступає не лише інструментом розвитку, але й фактором ризику, що вимагає додаткових досліджень у сфері безпеки.

Важливою особливістю ШІ є здатність працювати з масивними обсягами даних у реальному часі, знаходити приховані закономірності та створювати моделі для прогнозування майбутніх подій. На відміну від людини, яка має обмежені можливості опрацювання інформації, штучні інтелектуальні системи здатні аналізувати тисячі параметрів одночасно, що підвищує точність прийняття рішень. Оперативна адаптація моделей до змінних умов середовища дозволяє швидко реагувати на нові виклики та загрози.

Саме ці характеристики визначають цінність ШІ у сфері кібербезпеки. Інформаційне середовище постійно змінюється: з’являються нові типи атак, модифікуються інструменти зловмисників, збільшується кількість вразливостей у програмному забезпеченні. За таких умов час реакції на загрозу стає критично важливим фактором, а здатність ШІ миттєво аналізувати мережевий трафік, виявляти аномалії та прогнозувати розвиток подій забезпечує ефективність систем захисту. Це робить штучний інтелект одним із ключових напрямів розвитку сучасних технологій кіберзахисту.

На рис. 1.1 подано узагальнену схему, що відображає основні переваги застосування штучного інтелекту в інформаційній безпеці. Представлені аспекти демонструють широке коло можливостей, починаючи від обробки великих обсягів даних та мінімізації людських помилок і завершуючи автоматизованою відповіддю на підозрілу активність та прогнозуванням майбутніх атак.

Рис. 1.1 Переваги штучного інтелекту

Особливу увагу заслуговує здатність ШІ до адаптивності, яка забезпечує навчання на нових даних і швидку адаптацію методів захисту відповідно до змінних умов. Також важливим є компонент виявлення загроз, що охоплює як ідентифікацію незвичних патернів, так і виявлення нових форм атак, раніше невідомих традиційним системам. У поєднанні з безперервним моніторингом та

зменшенням суб'єктивного впливу людського фактора це створює фундамент для формування стійких, інтелектуально керованих систем кіберзахисту.

Методи машинного навчання (МН) формують основу більшості сучасних систем виявлення загроз. Основні підходи включають:

Навчання з учителем є одним із базових підходів у машинному навчанні, що передбачає використання навчального набору даних із наперед відомими правильними відповідями (мітками). Мета цього процесу полягає у тому, щоб алгоритм, отримуючи вхідні дані та приклади правильних результатів, поступово «навчався» встановлювати залежності між ознаками та відповідними класами або значеннями. Таким чином формується модель, здатна на основі нових, раніше невідомих даних робити прогнози чи класифікацію з певним рівнем точності. Даний підхід широко застосовується у задачах розпізнавання образів, аналізу текстів, прогнозування подій та, зокрема, у сфері інформаційної безпеки для виявлення аномалій та ідентифікації шкідливої активності в мережах.

На рис. 1.2 наведено архітектуру процесу навчання з учителем, яка наочно ілюструє взаємодію між вихідними сирими даними, навчальним набором і бажаним результатом.

Рис. 1.2 Архітектура навчання з учителем [6]

Центральним елементом процесу навчання з учителем є алгоритм, який під керівництвом наставника отримує доступ до маркованих прикладів і поступово

коригує свої параметри з урахуванням очікуваного результату. Вхідні дані проходять етап підготовки та обробки, у результаті чого модель набуває здатності перетворювати нову інформацію у прогнозовані виходи. Такий підхід забезпечує можливість класифікації чи групування даних на основі набутих знань, що робить його особливо ефективним для аналізу великих масивів інформації та дозволяє досягати високої точності і швидкості у прийнятті рішень.

Прикладами використання навчання з учителем у кібербезпеці є:

– виявлення вторгнень. На основі маркованих наборів даних, де окремі зразки позначені як «нормальний трафік» або «шкідлива активність», моделі навчаються розрізняти легітимні та підозрілі дії в мережі. Це дозволяє ідентифікувати атаки на ранніх етапах;

– класифікація шкідливого програмного забезпечення. Використання навчальних вибірок, у яких файли мають мітки «безпечний» чи «шкідливий», дає змогу алгоритмам точно визначати нові зразки вірусів, троянів чи шпигунських програм за їх характеристиками;

– фільтрація спаму та фішингових листів. Листи з позначками «спам/фішинг» або «безпечний» слугують основою для побудови моделей, які автоматично відсіюють небажані повідомлення та захищають користувачів від соціальної інженерії;

– аналіз поведінки користувачів. Завдяки навчальним даним, що містять приклади звичайних та аномальних дій користувачів, система здатна виявляти нетипову поведінку, яка може свідчити про компрометацію облікового запису чи внутрішню загрозу.

Навчання з учителем у сфері кібербезпеки забезпечує високу точність у класифікації загроз завдяки використанню попередньо маркованих даних. Цей підхід дозволяє будувати адаптивні системи, здатні виявляти широкий спектр атак – від мережевих вторгнень до соціальної інженерії. Його застосування сприяє зниженню ризиків та підвищенню рівня захищеності інформаційних систем, що підтверджує практичну доцільність інтеграції таких рішень у сучасні системи кіберзахисту.

Навчання без учителя використовується у випадках, коли дані не мають наперед визначених міток або правильних відповідей. Мета цього підходу полягає у виявленні прихованих структур, закономірностей чи групувань у даних. Алгоритм самостійно аналізує вхідну інформацію, виокремлює схожі об’єкти та формує кластери, не спираючись на заздалегідь визначені класи. Такий метод особливо корисний у тих ситуаціях, коли маркування даних є надто дорогим, тривалим або практично неможливим.

На рис. 1.3 зображено архітектуру процесу навчання без учителя. Схема демонструє, як вхідні «сирі» та нерозмічені дані проходять через етапи інтерпретації й обробки за допомогою алгоритму. У результаті відбувається автоматичне групування елементів за прихованими ознаками: подібні об’єкти потрапляють в один кластер, відмінні – в інший.

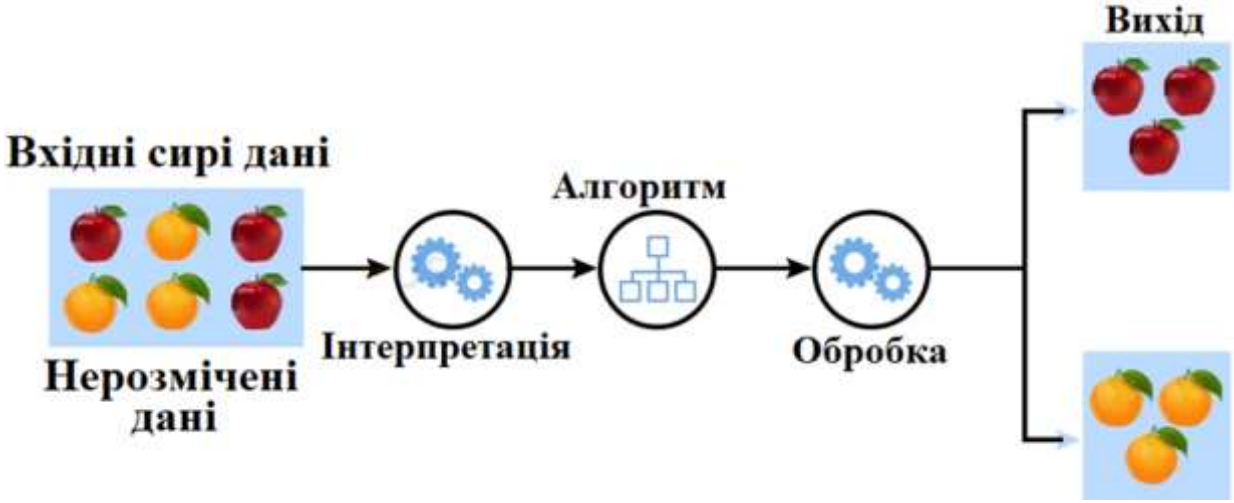

Рис. 1.3 Архітектура навчання без учителя [7]

Даний підхід дозволяє системі без участі людини виокремлювати закономірності, які можуть бути непомітними для аналітика. Це відкриває можливості для виявлення аномалій у мережевому трафіку, ідентифікації нових типів атак або групування невідомих зразків шкідливого програмного забезпечення, що значно підвищує ефективність систем кіберзахисту в умовах швидкоплинних змін кіберзагроз.

Приклади використання навчання без учителя у кібербезпеці включають:

– виявлення аномалій у мережевому трафіку. Кластеризація дозволяє виділити нетипові зразки поведінки, які не належать до жодного з відомих класів, що допомагає виявляти нові або раніше невідомі типи атак;

– групування зразків шкідливого програмного забезпечення. Алгоритми без учителя об’єднують нові програми за схожими характеристиками, навіть якщо вони ще не класифіковані у базах даних антивірусних систем;

– виявлення внутрішніх загроз. Аналіз поведінкових даних співробітників дозволяє виокремити підозрілі патерни без необхідності їх попереднього маркування, що знижує ризики зловживань усередині організації;

– сегментація мережевих пристроїв. Автоматичне групування обладнання за типом або поведінковими характеристиками спрощує моніторинг, виявлення аномалій та формування політик безпеки.

Отже, навчання без учителя дозволяє працювати з немаркованими даними, виявляти приховані закономірності та ідентифікувати загрози, які ще не мають відомих ознак. Використання цього підходу сприяє побудові більш адаптивних систем захисту, здатних реагувати на нові виклики у динамічному інформаційному середовищі.

Навчання з підкріпленням є підходом у машинному навчанні, який базується на взаємодії агента з навколишнім середовищем. У процесі навчання агент виконує дії, отримує від середовища зворотний зв’язок у вигляді винагороди або покарання та поступово формує стратегію поведінки, яка максимізує сумарний виграш. Основна мета такого підходу полягає не лише у знаходженні правильної відповіді, а й у виробленні оптимальної послідовності рішень у динамічних і часто невизначених умовах. Цей метод широко застосовується у сфері робототехніки, управління ресурсами, автономних систем і, зокрема, у кібербезпеці для розробки адаптивних стратегій реагування на загрози.

На рис. 1.4 подано архітектуру процесу навчання з підкріпленням. Вона демонструє, як вхідні сирі дані надходять у середовище, де агент аналізує ситуацію та здійснює вибір дій. Середовище, у свою чергу, генерує зворотний зв’язок, що використовується агентом для вдосконалення його майбутніх рішень.

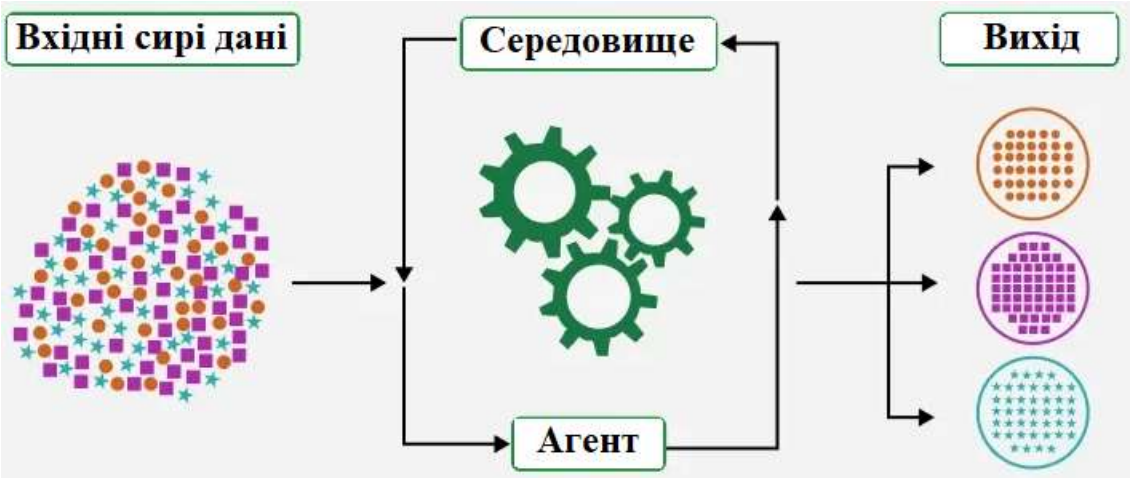

Рис. 1.4 Архітектура навчання з підкріпленням [8]

Схема відображає циклічний характер навчання: агент постійно взаємодіє з середовищем, коригуючи свою поведінку залежно від отриманих результатів. Це дозволяє системам штучного інтелекту працювати в умовах невизначеності, пристосовуватися до змінних факторів і формувати оптимальні стратегії для досягнення поставлених цілей.

Яскравими прикладами використання навчання з підкріпленням у кібербезпеці є:

– адаптивне виявлення вторгнень. Агент формує стратегії реагування на підозрілу активність у мережі, поступово навчаючись блокувати шкідливі дії та мінімізувати кількість хибних спрацювань;

– оптимізація політик доступу. Алгоритми з підкріпленням використовуються для динамічного налаштування правил авторизації та автентифікації, що дозволяє знизити ризики компрометації облікових записів;

– захист від DDoS-атак. Агент навчається визначати ефективні дії для зниження навантаження на систему, наприклад перенаправлення трафіку або активацію резервних серверів;

– автоматизоване реагування на інциденти. Системи на основі цього підходу можуть виробляти оптимальні стратегії для ізоляції заражених вузлів, відновлення працездатності мережі чи розподілу ресурсів під час атаки.

Навчання з підкріпленням у сфері кібербезпеки відкриває можливість створення гнучких і адаптивних систем захисту, здатних самостійно виробляти

ефективні стратегії дій у реальному часі. Цей підхід дозволяє не лише виявляти загрози, але й активно протидіяти їм, забезпечуючи стійкість інформаційних систем до сучасних і майбутніх кібератак.

Широке застосування ШІ створює новий простір для нападників. Дослідження Неретіна та Харченка аналізує кібербезпеку систем штучного інтелекту. Автори класифікували потенційні атаки на платформи, алгоритми та дані і виявили, що найвищі ризики становлять змагальні атаки (adversarial attacks) та атаки отруєння даних, для яких контрзаходи ще не розвинуті [9]. Інші небезпечні вектори включають модифікацію даних, відмову в обслуговуванні, витік даних, «троянські» атаки та крадіжку моделей. Неретін і Харченко підкреслюють, що розроблення безпечних систем ШІ потребує формалізації життєвого циклу та стандартів кіберзахисту, зокрема для сервісів AI-as-a-Service.

Серед ключових контрзаходів, які пропонують дослідники:

– захист навчальних даних та контроль доступу. Необхідно забезпечити цілісність навчальних наборів і захищати їх від стороннього втручання (наприклад, шляхом криптографічного контролю сум);

– розроблення моделей, стійких до змагальних атак. Методи для підвищення стійкості включають adversarial training (навчання на модифікованих прикладах) та використання моделей із сертифікованим захистом;

– моніторинг і валідація моделей. Під час експлуатації необхідно регулярно перевіряти результати моделі на наявність відхилень та проводити ротацію моделей;

– прозорість і можливість аудиту. Використання explainable AI (XAI) допомагає розуміти, як моделі приймають рішення, що полегшує виявлення аномалій у поведінці.

Сучасні методи кіберзахисту часто поєднують традиційні механізми безпеки з принципами «архітектури нульової довіри» (Zero Trust). Ця модель ґрунтується на припущенні, що система вже скомпрометована, тому нікому не можна беззастережно довіряти. Доступ надається лише після ретельної перевірки та за принципом найменших привілеїв. Як наголошується в аналітичній статті, Zero

Trust вимагає підтвердження кожного доступу, профілювання поведінки користувачів та мінімізації привілеїв, що суттєво ускладнює роботу зловмисників [10]. Розгортання Zero Trust разом із UBA та SIEM дозволяє будувати гнучкі та стійкі системи захисту.

1.2 Дослідження основних типів загроз та вразливостей інформаційних систем

Кіберзагрози можна визначити як навмисні дії, спрямовані на пошкодження або викрадення даних, порушення роботи систем чи іншої цифрової інфраструктури. За матеріалами блогу DataLabs, кіберзагроза включає витік даних, комп’ютерні віруси, атаки відмови в обслуговуванні (DDoS) та інші типи атак [11]. Джерела загроз можуть бути зовнішніми або внутрішніми: державні та терористичні організації, корпоративні шпигуни, організована злочинність, хактивісти, незадоволені інсайдери, природні катастрофи та випадкові помилки користувачів. Важливо розуміти як тип, так і джерело загрози, щоб розробити відповідні засоби захисту.

Здобутий досвід Української CERT-UA демонструє зростання агресивності кіберзлочинців. У 2023 році зафіксовано тисячі атак на державні й приватні структури: 347 кібератак проти уряду та держорганів, 276 проти органів місцевого самоврядування, 175 у сфері безпеки й оборони, 127 проти комерційних організацій, десятки атак на енергетичний, телекомунікаційний та освітній сектори [10]. Прогноз на 2024 рік передбачає зростання кількості атак на 15 % і значне зростання атак з метою викрадення конфіденційних даних. Прогнозується, що кількість DDoS-атак може сягнути 2 млн випадків. Нападники використовують штучний інтелект для персоналізації фішингових листів, що робить атаки менш помітними.

Сучасний кіберпростір характеризується не лише кількісним зростанням атак, а й їх якісним ускладненням, зокрема використанням штучного інтелекту для маскування та підвищення ефективності злочинних дій. Це обумовлює

необхідність систематизації та аналізу основних типів кібератак, які становлять найбільшу загрозу для інформаційної безпеки:

Атаки типу DoS (Denial of Service) та DDoS (Distributed Denial of Service) спрямовані на порушення доступності інформаційних ресурсів – веб-сайтів, серверів, мережевих сервісів або відмовостійких компонентів інфраструктури. У класичному DoS-випадку зловмисник генерує надмірну кількість запитів або спеціально сформований трафік безпосередньо до цілі, що призводить до виснаження обчислювальних, пам’ятних або мережевих ресурсів і, як наслідок, до відмови у наданні послуг легітимним користувачам. У DDoS-атакі аналогічна дія реалізується розподілено – через велику кількість контрольованих зловмисником машин (ботнет), що робить атаку значно масштабнішою, стійкішою до блокування та складнішою для локалізації. Мотиви проведення таких атак варіюються від прагнення створити тимчасові перебої в роботі сервісу (вандалізм, тиск, конкурентні або економічні мотиви) до використання DDoS як прикриття для паралельних атак (викрадення даних, проникнення) або як частини складних кібероперацій.

На рис. 1.5 наведено схему типової DDoS атаки: зловмисник керує сервером команд і керування, який координує велику кількість інфікованих пристроїв (ботів). Кожен бот надсилає до жертви численні запити або спеціалізований шкідливий трафік, що в сукупності створює ефект перевантаження цільового ресурсу.

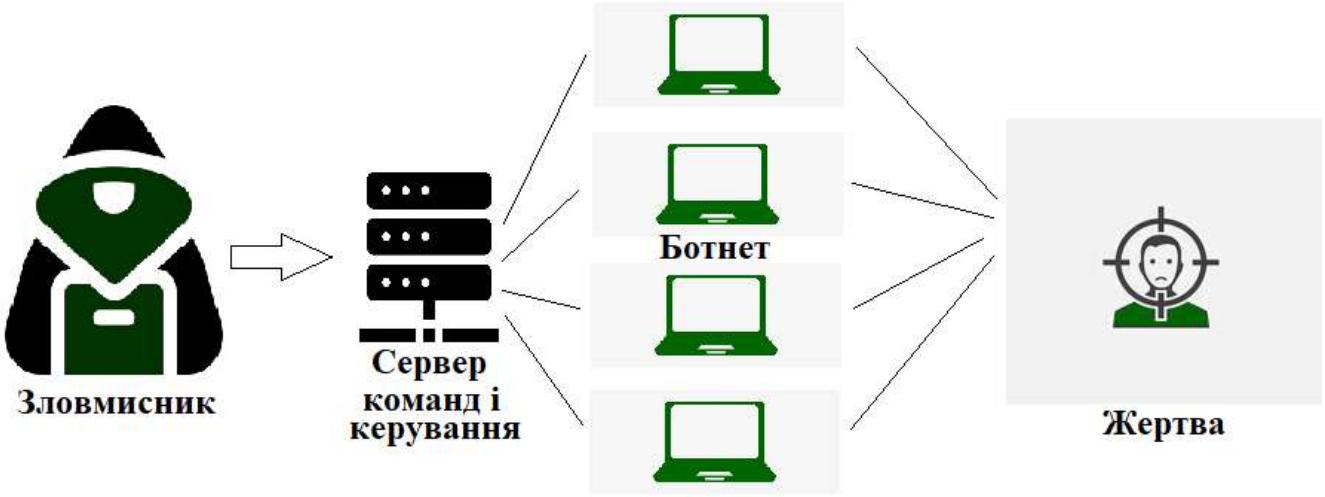

Рис. 1.5 Схема DDoS атаки [12]

Ілюстрація чітко відтворює головні складові загрозної архітектури: ініціатора атаки, інфраструктуру керування та розподілену мережу виконавців-ботів, що спрямовують потоки на обрану ціль. Така організація дозволяє зловмисникам збільшити інтенсивність атаки, приховати джерело ініціації через багато проміжних вузлів та застосовувати різні техніки (наприклад, amplification/reflection, SYN-flood, HTTP-flood) для підвищення ефективності. У практичному аспекті це підкреслює необхідність комплексних заходів захисту: фільтрації трафіку на крайових і магістральних елементах мережі, використання систем виявлення аномалій, розподілених систем балансування навантаження та співпраці з провайдерами для швидкої ізоляції шкідливих потоків.

Фішинг – це метод соціальної інженерії, спрямований на отримання конфіденційної інформації шляхом маскування шкідливих повідомлень під легітимні комунікації. Зловмисники формують повідомлення, що імітують офіційні листи банків, сервісів або колег, і спрямовують їх на потенційні жертви з метою спонукати до переходу за посиланням або відкриття вкладення, яке запускає процес компрометації. Основну уразливість при цьому становить людський фактор – довіра до зовнішніх ознак правдоподібності та швидка реакція на інформацію, що нібито вимагає негайних дій.

На рис. 1.6 ілюстровано типовий сценарій фішингової атаки: зловмисник готує і надсилає електронне повідомлення, що маскується під офіційну комунікацію від імені відомого сервісу або довіреної особи, після чого одержувач, переходячи за вбудованим посиланням, потрапляє на підроблений веб-ресурс, зовні майже не відмінний від оригіналу. Під час взаємодії з такою фальшивою сторінкою жертва вводить свої облікові дані, які в режимі реального часу потрапляють до нападника; далі отримана інформація використовується для несанкціонованого доступу до легітимних сервісів потерпілого або для розгортання подальших етапів атаки, зокрема компрометації контактів і розсилання нових фішингових повідомлень від імені постраждалої особи.

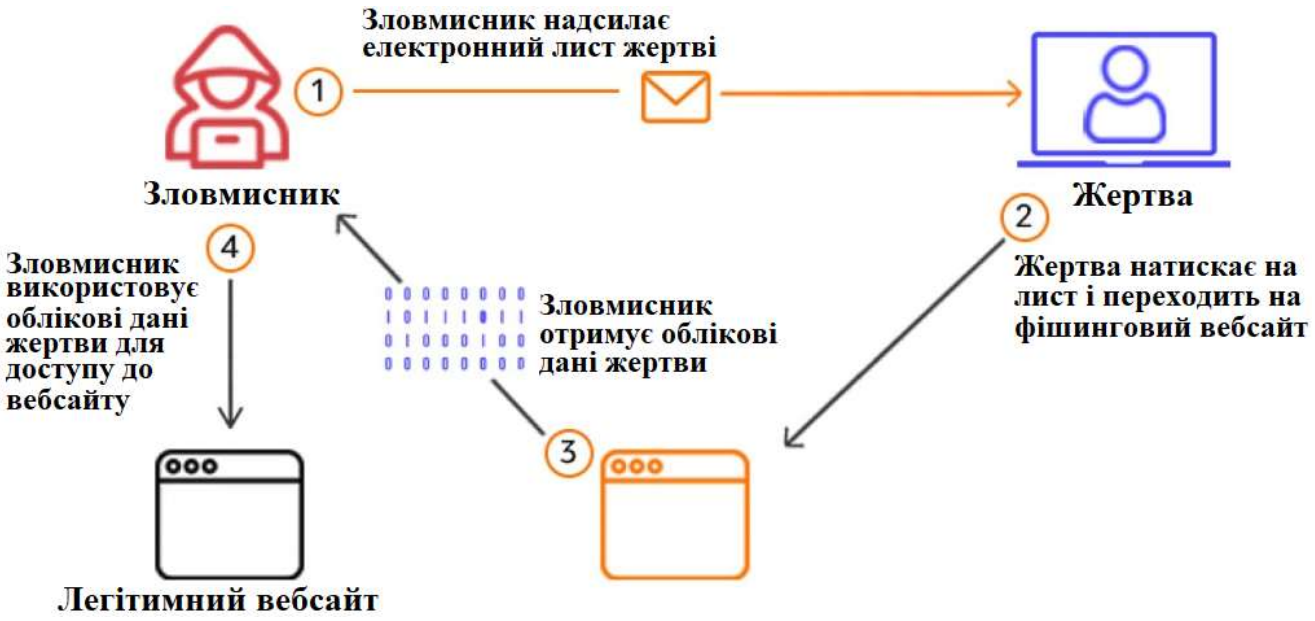

Рис. 1.6 Схема фішингової атаки [13]

До фішингу належать також інші методи соціальної інженерії, що застосовують різні канали та підходи:

– Spear-phishing – цілеспрямований фішинг проти конкретних осіб або організацій, що базується на попередньому зборі інформації про жертву;

– Vishing (голосовий фішинг) – використання телефонних дзвінків для виманювання секретних даних або підбурювання до дій;

– Smishing – фішингові повідомлення через SMS;

– Pretexting – створення правдоподібного вигаданого сценарію (роль представника сервісу, інспектора тощо) для отримання довірливої інформації;

– Baiting – розміщення «спокусливого» носія або файлу (наприклад, USB-накопичувача) з метою його підняття жертвою та запуску шкідливого коду;

– Tailgating (piggybacking) – фізичне проникнення до захищених приміщень шляхом супроводу за уповноваженою особою;

– Business Email Compromise (BEC) – компрометація корпоративної електронної пошти з метою ініціювання фінансових чи правових операцій від імені керівництва.

Соціальна інженерія залишається одним із найпоширеніших і найуспішніших векторів атак через свою орієнтацію на людську поведінку; фішинг служить базовим інструментом для отримання початкового доступу або викрадення

облікових даних. Ефективний захист вимагає комбінованого підходу: технічних механізмів (фільтрація пошти, DMARC/SPF/DKIM, багатоетапна автентифікація, ізоляція виконання контенту), регулярного навчання персоналу і моделювання реальних атак, а також відпрацьованих процедур реагування на інциденти, що дозволяють швидко локалізувати та нейтралізувати наслідки компрометації.

Атаки на ланцюг постачання належать до одного з найнебезпечніших типів сучасних кіберзагроз, оскільки вони спрямовані не безпосередньо на кінцеву організацію, а на її постачальників програмного чи апаратного забезпечення. Суть цих атак полягає у компрометації процесів розробки, збирання або оновлення продуктів, унаслідок чого до легітимного програмного коду впроваджується шкідливий компонент. Таким чином, зловмисники отримують можливість поширювати власне шкідливе програмне забезпечення разом із офіційними оновленнями або дистрибутивами, що суттєво ускладнює виявлення атаки та створює ефект «ланцюгової реакції» ураження значної кількості організацій. Основною метою подібних атак є отримання доступу до критично важливих систем і даних, шпигунство, саботаж або створення прихованих каналів управління.

На рис. 1.7 зображено схему компрометації ланцюга постачання.

Рис. 1.7 Схема атаки через компрометацію ланцюга постачання [14]

Схема відтворює послідовність дій зловмисника, що починається з непомітного впровадження шкідливого коду у легітимний вихідний код на етапі збірки програмного забезпечення; надалі інфіковані збірки потрапляють на сервери постачальника і розповсюджуються як офіційні оновлення або дистрибутиви; отримавши інфіковане оновлення, клієнтські системи автоматично інсталюють шкідливі компоненти, внаслідок чого зловмисники здобувають віддалений доступ до систем жертв і можливість виконувати подальші дії – від викрадення конфіденційних даних до встановлення стійкого контролю над інфікованими машинами.

Саме описані вище загрози природно переходять у сучасний клас атак, що використовують можливості штучного інтелекту та машинного навчання як інструменту підвищення масштабності, витонченості та автоматизації шкідливих дій. Зловмисники застосовують моделі генерації контенту для створення персоналізованих фішингових повідомлень, використовують нейромережі для автоматичної генерації фальшивих голосових чи відеоматеріалів, а також експлуатують методи автоматичного пошуку вразливостей і оптимізації поведінки шкідливого ПЗ. У відповідь на це захисні системи змушені поєднувати методи традиційного кіберзахисту з алгоритмами ШІ для аналізу поведінкових патернів, виявлення аномалій і прогнозування розвитку інцидентів.

Нижче наведено табл. 1.1 з прикладами різних методів атак із використанням ШІ, їхніми характеристиками та типовими заходами протидії.

Таблиця 1.1

Методи атак із використанням штучного інтелекту

Метод атаки	Опис	Використана техніка	Потенційна шкода
1	2	3	4
Генеративний фішинг (AI-phishing)	Масова генерація персоналізованих фішингових листів і повідомлень, адаптованих під конкретну жертву	Генеративні моделі, NLG з донавчанням на публічних даних	Витік облікових даних, компрометація акаунтів, фінансові втрати

Продовження таблиці 1.1.

1	2	3	4
Deepfake-соціальна інженерія	Створення правдоподібних аудіо/відео матеріалів для маніпуляцій із персоналом або клієнтами	GAN, трансформери для генерації аудіо/відео	Шахрайство, витік конфіденційної інформації, підрив довіри
Автоматичний пошук вразливостей	Використання ML для автоматичного сканування коду й знаходження вразливостей у ПО	Моделі для статичного/динамічного аналізу, NLP для коду	Прискорене виявлення «нульденних» вразливостей, масштабні експлойти
AI-кероване шкідливе ПЗ	Підлаштовуване шкідливе ПЗ, що змінює поведінку для обходу захисту й максимізації ефекту	Рекомендаційні політики, еволюційні алгоритми, онлайн-навчання	Тривала компрометація, витік даних, складність виявлення

Отже, розвиток методів ШІ створює нові класи загроз, які інтегрують автоматизацію, генерацію обманного контенту та адаптивну поведінку шкідливого коду; проте такий же набір методів і підходів може бути застосований для посилення заходів захисту, що відкриває шлях до розвитку контрзаходів на перетині кібербезпеки й машинного навчання, стандартів оцінювання ризиків і процедур відповідальності.

Внутрішні загрози становлять особливу категорію ризиків для інформаційної безпеки, оскільки походять зсередини організації і використовують наявні привілеї, знання бізнес-процесів та доступ до конфіденційних ресурсів. Такі загрози можуть бути умисними (диктовані фінансовою вигодою, шкідливою мотивацією або саботажем) або ненавмисними (результат неуважності, помилкових дій чи недостатньої кваліфікації). Відмінною рисою внутрішніх атак є їхня прихованість і здатність завдавати значної шкоди за короткий час; тому протидія їм потребує поєднання технічних механізмів контролю, організаційних процедур та постійного моніторингу поведінки користувачів.

В табл. 1.2 наведено види з типовими видами внутрішніх загроз, їх характеристикою, індикаторами та прикладними заходами протидії.

Таблиця 1.2

Види внутрішніх загроз

Тип загрози	Характеристика	Типові індикатори	Потенційні наслідки
Злочинний інсайдер	Усвідомлене й навмисне використання доступу для викрадення даних або шкоди системам	Незвичні запити доступу, експорт великих обсягів даних, робота у нетиповий час	Витік конфіденційної інформації, фінансові збитки, репутаційні втрати
Недбалість / помилкові дії	Неправильні операції або порушення процедур через помилку чи низьку кваліфікацію	Часті помилки в конфігурації, неправильні зміни у системах, випадкове видалення файлів	Порушення доступності сервісів, витік даних, експлуатаційні збої
Зловживання привілеями адміністраторів	Неправомірне або невиправдане використання прав адміністратора для зміни конфігурацій або доступу до даних	Перевищення прав, невиправдані зміни у налаштуваннях, відсутність доказів необхідності	Стійке порушення цілісності систем, важкість відновлення нормальної роботи
Ризики від третіх сторін (підрядники)	Компрометація або недбалість у постачальників, що мають доступ до систем	Незвичні оновлення, сторонні підключення, невідповідні права у зовнішніх акаунтів	Вторгнення через довірені канали, ланцюгова компрометація

Отже, внутрішні загрози становлять одну з найбільш небезпечних категорій ризиків для інформаційної безпеки, оскільки вони поєднують у собі прихований характер, високий рівень доступу до критичних ресурсів та знання внутрішніх процесів організації. Їхня особливість полягає в тому, що вони можуть завдавати шкоди значно швидше та масштабніше, ніж зовнішні атаки, а їх наслідки охоплюють фінансові втрати, репутаційні ризики та порушення стабільності функціонування інформаційних систем.

Окрім внутрішніх і зовнішніх кібератак, важливим чинником, що впливає на стан інформаційної безпеки, залишаються фізичні та природні загрози. Вони безпосередньо пов’язані з матеріальною інфраструктурою організації та середовищем її функціонування. На відміну від класичних кіберзагроз, їхня природа полягає у фізичному впливі на обладнання, носії інформації або канали зв’язку, що може призвести до часткової чи повної втрати доступності, цілісності та збереженості даних. Сюди належать як стихійні лиха (повені, пожежі, землетруси), так і техногенні чи навмисні дії (крадіжки, вандалізм, пошкодження обладнання).

Нижче наведено табл. 1.3, яка узагальнює основні види фізичних і природних загроз, їхні характеристики та можливі наслідки для інформаційних систем.

Таблиця 1.3

Фізичні та природні загрози інформаційній безпеці

Тип загрози	Характеристика	Типові індикатори	Потенційні наслідки
Пожежа	Займання в приміщеннях або серверних залах	Дим, різке підвищення температури, спрацювання пожежної сигналізації	Знищення обладнання, втрата даних, тривала зупинка бізнес-процесів
Повінь/ затоплення	Природні явища чи аварії водопостачання	Підняття рівня води, пошкодження кабельних комунікацій	Корозія техніки, втрата носіїв даних, вихід з ладу інфраструктури
Крадіжка обладнання	Викрадення носіїв даних або технічних засобів	Відсутність обладнання, сліди злому, втручання у приміщення	Несанкціонований доступ до даних, фінансові збитки, порушення роботи систем
Вандалізм чи саботаж	Пошкодження інфраструктури чи носіїв інформації	Зламані двері, кабелі, механічні пошкодження	Порушення функціонування систем, компрометація або втрата даних
Землетрус чи інше стихійне лихо	Непередбачувані природні явища	Перебої у живленні, руйнування інфраструктури	Втрати обладнання, знищення інформації, повне припинення роботи

Фізичні та природні загрози вирізняються непередбачуваним характером та масштабністю впливу, здатністю паралізувати інформаційну інфраструктуру й завдавати критичної шкоди бізнес-процесам, що робить їх однією з ключових категорій ризиків у сфері інформаційної безпеки.

1.3 Аналіз існуючих інструментів та програмних рішень для автоматизованого виявлення загроз

У сучасних умовах зростаючої інтенсивності та складності кібератак традиційні підходи до забезпечення інформаційної безпеки вже не здатні гарантувати належний рівень захисту. Це зумовлює необхідність впровадження інтелектуальних інструментів і програмних рішень, здатних не лише фіксувати факти порушень, а й здійснювати автоматизований аналіз подій, виявляти приховані закономірності та оперативно реагувати на загрози.

На рис. 1.8 подано класифікацію основних інструментів виявлення та реагування на загрози, які інтегрують можливості збору, аналізу та автоматизації заходів кіберзахисту.

Рис. 1.8 Інструменти виявлення та реагування на загрози

Сучасна архітектура засобів безпеки охоплює кілька ключових напрямів: системи IDS/IPS для моніторингу та блокування атак, SIEM-рішення для централізованого збору і кореляції подій, EDR/EPP для поведінкового аналізу на кінцевих точках, XDR для комплексного відстеження загроз у мережі та хмарних середовищах, а також платформи SOAR і Threat Intelligence, які забезпечують автоматизацію реагування та оновлення баз знань. У сукупності ці інструменти формують єдину екосистему, що дозволяє організаціям своєчасно виявляти інциденти, будувати сценарії реагування і підвищувати рівень стійкості до кібератак.

Розглядаючи різноманіття інструментів і платформ, які застосовуються у сфері інформаційної безпеки, доцільно звернути увагу на конкретні рішення, що набули найбільшого поширення у практиці захисту корпоративних мереж. Однією з таких систем є Snort, яка використовується як система виявлення та запобігання вторгненням (рис. 1.9). Її основне призначення полягає у моніторингу мережевого трафіку в реальному часі, виявленні підозрілої активності, аналізі мережевих пакетів і своєчасному реагуванні на потенційні атаки.

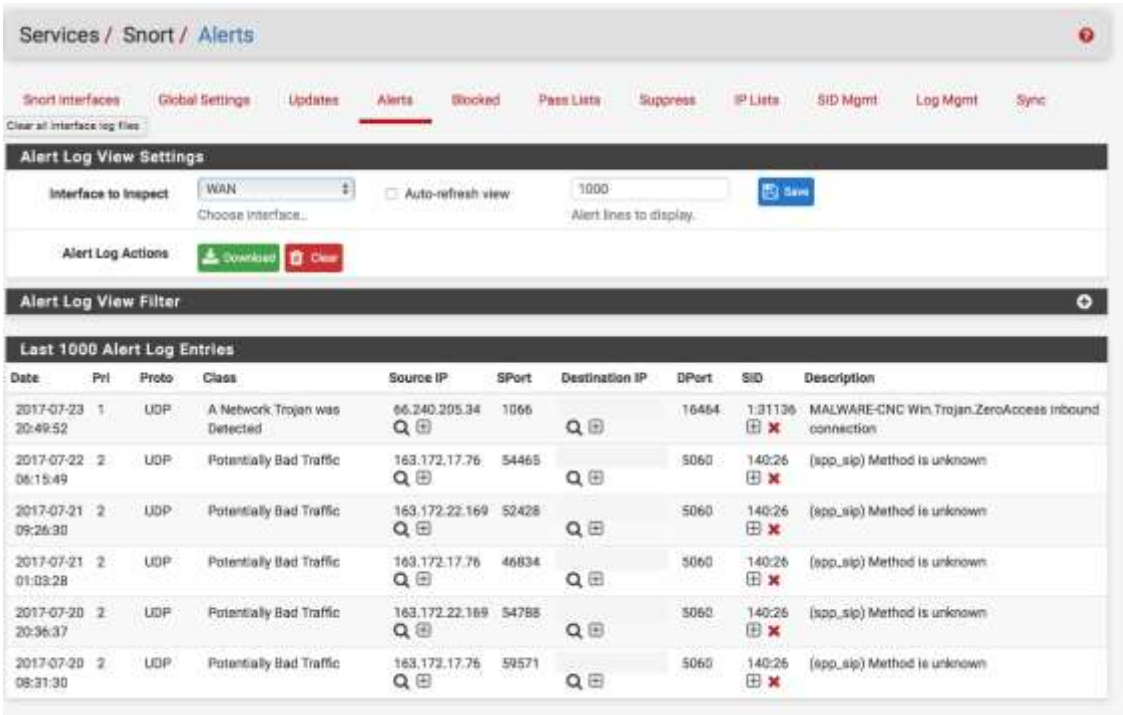

Рис. 1.9 Приклад інтерфейсу для роботи із Snort [15]

Архітектура Snort побудована модульно і включає кілька взаємопов’язаних компонентів. На початковому етапі дані потрапляють до модуля захоплення трафіку, де відбувається фільтрація пакетів. Далі вони передаються до системи декодування, яка визначає протоколи та структуру отриманих даних. Ключову роль відіграє механізм аналізу правил, що застосовує базу сигнатур і дозволяє виявляти як відомі атаки, так і підозрілі шаблони. Результати обробки спрямовуються до модуля логування та оповіщення, який фіксує події та надсилає повідомлення адміністратору. Завдяки такій архітектурі Snort забезпечує гнучкість у налаштуванні та можливість інтеграції з іншими засобами кіберзахисту.

Переваги Snort:

– відкритий вихідний код і безкоштовна ліцензія, що робить систему доступною для широкого кола користувачів [16];

– гнучка модульна архітектура з можливістю налаштування правил під специфічні потреби організації;

– велика спільнота розробників і користувачів, яка регулярно оновлює базу сигнатур та забезпечує швидке реагування на нові загрози [17];

– можливість інтеграції з іншими системами безпеки та SIEM-рішеннями для підвищення ефективності моніторингу.

Недоліки Snort:

– високі вимоги до ресурсів при великому обсязі трафіку, що може знижувати продуктивність у масштабних мережах;

– орієнтація здебільшого на сигнатурний аналіз, через що складно виявляти нові або модифіковані атаки без попередньо створених правил [18];

– складність налаштування для недосвідчених користувачів, що потребує значних технічних знань [19];

– велика кількість хибних спрацювань у разі недостатньо якісного налаштування правил.

Отже, Snort є одним із найбільш відомих і широко застосовуваних інструментів для виявлення вторгнень, який поєднує доступність, потужний

функціонал і активну підтримку спільноти. Попри певні обмеження, пов’язані з продуктивністю та складністю налаштування, він залишається ефективним рішенням для організацій, що прагнуть підвищити рівень безпеки своїх інформаційних систем.

Suricata є сучасною системою виявлення та запобігання вторгненням, яка поєднує можливості IDS, IPS та моніторингу мережевого трафіку (рис. 1.10). Її основне призначення полягає у виявленні загроз у реальному часі, аналізі протоколів, детекції аномалій і фіксації повного мережевого трафіку для подальшого дослідження інцидентів. На відміну від багатьох інших рішень, Suricata підтримує багатопотокову обробку, що забезпечує високу продуктивність у мережах із великим навантаженням.

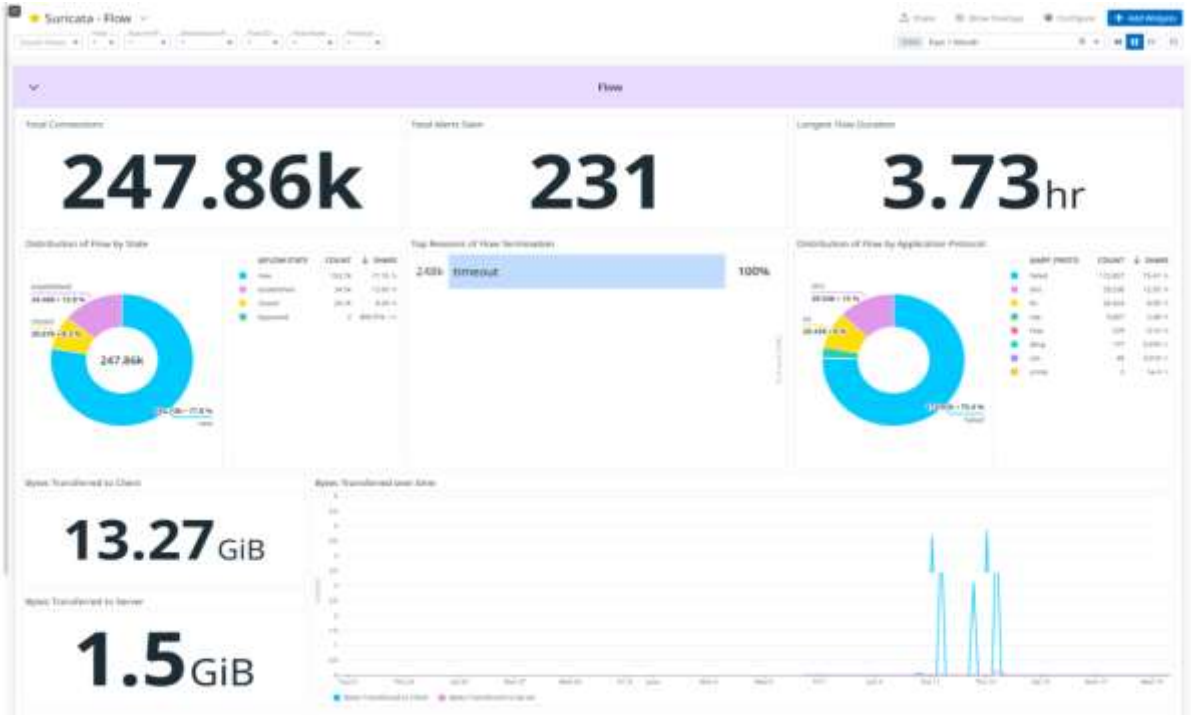

Рис. 1.10 Приклад інтерфейсу системи Suricata [20]

Архітектура Suricata базується на поєднанні сигнатурного та поведінкового аналізу. Вона використовує ядро для швидкої обробки пакетів, яке розподіляє навантаження між кількома потоками, дозволяючи системі масштабуватися та працювати ефективно навіть у розподілених середовищах. Додатково інтегровано механізми підтримки правил Snort, глибокий аналіз протоколів прикладного рівня та можливість експорту даних у зовнішні системи SIEM і аналітичні платформи.

Завдяки такій архітектурі Suricata забезпечує гнучкість і адаптивність у виявленні складних багаторівневих атак.

Переваги Suricata:

– підтримка багатопотокової обробки, що забезпечує високу продуктивність у великих мережах [21];

– можливість використання правил Snort, що спрощує інтеграцію з існуючими рішеннями;

– поглиблений аналіз мережевих протоколів прикладного рівня [22];

– інтеграція з SIEM-системами та підтримка різних форматів журналів для зручної аналітики.

Недоліки Suricata:

– високі вимоги до апаратних ресурсів у випадку великих обсягів трафіку;

– складність налаштування та адміністрування для недосвідчених користувачів [23];

– значний обсяг даних у логах, що ускладнює їх обробку без додаткових інструментів;

– обмежені можливості у виявленні загроз, які не мають чітких сигнатур або відомих патернів.

Отже, Suricata є потужним і гнучким інструментом для виявлення та запобігання вторгненням, який поєднує високу продуктивність із широким спектром функцій аналізу. Вона ефективно застосовується у великих корпоративних та державних мережах, забезпечуючи якісний моніторинг і контроль безпеки, однак потребує належних ресурсів і професійного адміністрування.

Zeek – це мережева система моніторингу безпеки, яка орієнтована не лише на виявлення вторгнень, а й на глибокий аналіз трафіку (рис. 1.11). Його призначення полягає у зборі детальної інформації про мережеві з’єднання, ідентифікації аномалій у поведінці користувачів чи сервісів, формуванні журналів подій і наданні гнучких можливостей для написання власних сценаріїв реагування.

Архітектура Zeek базується на двох основних компонентах: механізмі низькорівневого захоплення трафіку, що забезпечує високопродуктивну обробку даних у реальному часі, та інтерпретаторі сценаріїв, який дає змогу користувачам створювати специфічні політики для аналізу й класифікації подій. Такий підхід робить систему гнучкою та придатною для адаптації до різних середовищ, дозволяючи інтегрувати її з іншими інструментами кіберзахисту.

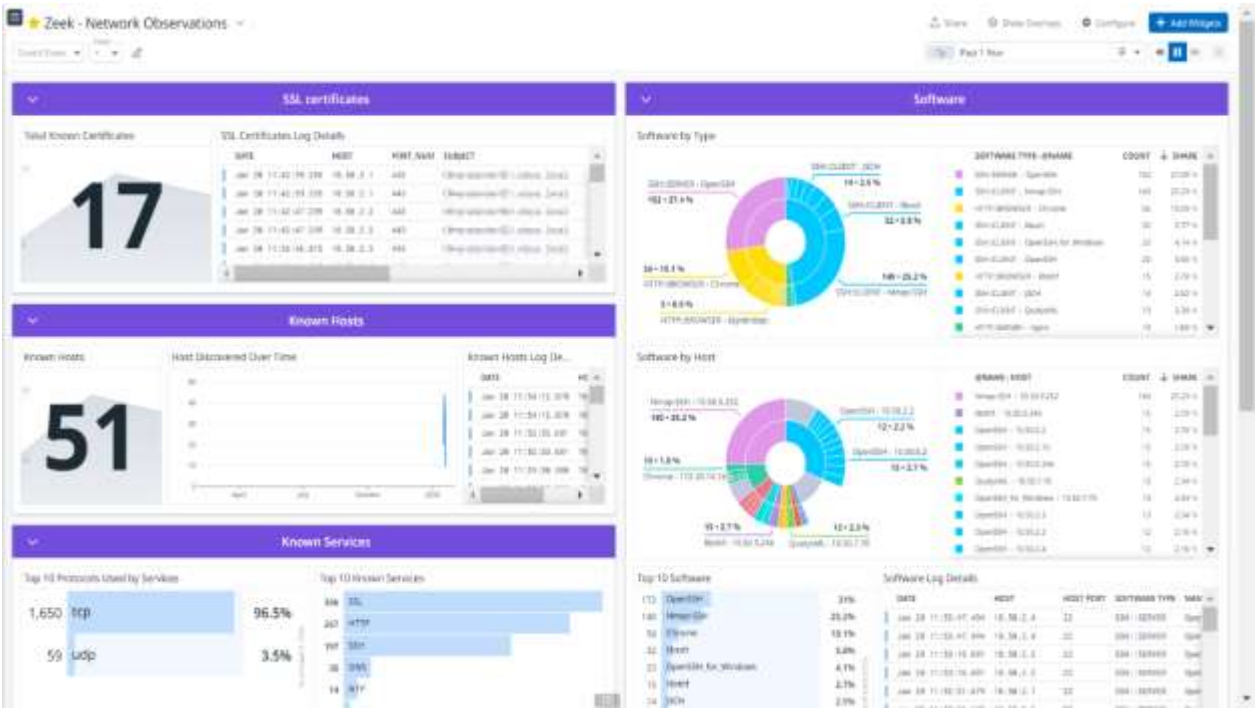

Рис. 1.11 Приклад інтерфейсу системи Zeek [24]

Переваги Zeek:

– глибокий аналіз мережевого трафіку з можливістю створення детальних логів і сценаріїв [25];

– висока гнучкість завдяки власній мові сценаріїв, що дозволяє адаптувати систему під конкретні завдання;

– легка інтеграція з іншими інструментами кіберзахисту (SIEM, IDS/IPS, системами аналізу логів) [26];

– активна спільнота та регулярні оновлення, що забезпечують актуальність у боротьбі з новими загрозами.

Недоліки Zeek:

– високі вимоги до обчислювальних ресурсів при обробці великого обсягу трафіку [26];

– складність налаштування та потреба у спеціалізованих знаннях для ефективного використання;

– обмежені можливості у виявленні загроз у зашифрованому трафіку (наприклад, HTTPS);

– відсутність автоматичного блокування атак – система здебільшого виконує роль моніторингу й аналізу [28].

Zeek є потужним інструментом для детального аналізу мережевої активності та виявлення аномалій, який надає організаціям глибокі знання про власну інфраструктуру. Попри вимогливість до ресурсів і складність налаштування, його використання виправдане у середовищах, де потрібен всебічний контроль за трафіком і високий рівень прозорості мережевих процесів.

Таким чином, проведений аналіз інструментів і програмних рішень для автоматизованого виявлення загроз показав, що класичні системи IDS/IPS, попри свою ефективність, мають низку обмежень, зокрема у виявленні нових та складних атак, а також у роботі з великими обсягами даних. Це визначає необхідність пошуку нових підходів, серед яких особливе місце посідають методи штучного інтелекту [27]. Їх застосування відкриває можливості для підвищення точності класифікації загроз, зниження кількості хибних спрацювань, адаптації до мінливих сценаріїв атак та формування систем активного реагування у реальному часі. Подальше дослідження в межах даної роботи буде зосереджене саме на розробці та оцінюванні таких підходів, що дозволить сформувати практичні рекомендації щодо інтеграції методів ШІ в сучасні архітектури кіберзахисту.

1.4 Формулювання завдань дослідження та постановка проблеми

Сучасний стан розвитку інформаційного суспільства супроводжується стрімким зростанням кількості кіберзагроз, які відзначаються високим рівнем складності та динамічності. Традиційні системи захисту, що ґрунтуються на сигнатурному або статичному аналізі, демонструють обмежену здатність до

виявлення нових, невідомих атак, а також до оперативної адаптації у мінливих умовах. Така ситуація формує науково-практичну проблему – відсутність універсальних та достатньо гнучких методів, здатних забезпечити своєчасне виявлення та запобігання інцидентам безпеки у масштабних та різнорідних інформаційних системах.

Водночас широке поширення технологій штучного інтелекту створює передумови для нового етапу розвитку методів кіберзахисту. Алгоритми машинного навчання, нейронні мережі та інші інтелектуальні підходи демонструють здатність до обробки великих масивів даних, виявлення прихованих закономірностей та формування прогнозів щодо потенційних загроз. Проте їх застосування у сфері інформаційної безпеки стикається з низкою викликів: нестачею якісно маркованих даних для навчання, складністю інтерпретації рішень моделей, високими обчислювальними витратами, а також ризиками упередженості та зниження точності у реальних умовах.

Таким чином, постає проблема розробки науково обґрунтованих підходів до інтеграції методів штучного інтелекту у системи виявлення й протидії кіберзагрозам, які враховували б не лише їх потенціал, а й практичні обмеження. Для вирішення поставленої проблеми необхідно провести ґрунтовний аналіз існуючих методів та інструментів, визначити ключові переваги й недоліки, сформувати вимоги до нових підходів, а також здійснити експериментальне дослідження ефективності алгоритмів у задачах класифікації та прогнозування атак.

Відповідно до мети дослідження, яка полягає у розробці рекомендацій щодо використання технологій штучного інтелекту в інформаційній безпеці, необхідно вирішити такі основні завдання:

– провести аналіз сучасних методів і підходів застосування штучного інтелекту у сфері кіберзахисту;

– дослідити основні типи загроз і вразливостей інформаційних систем з метою визначення їхньої специфіки для навчання інтелектуальних моделей;

– виконати порівняльну оцінку існуючих інструментів та програмних рішень для автоматизованого виявлення атак;

– розробити методичні основи застосування алгоритмів машинного навчання в інформаційній безпеці, включаючи математичні моделі, критерії оцінювання та архітектуру програмної реалізації;

– реалізувати експериментальне дослідження, спрямоване на перевірку працездатності обраних методів у симульованому середовищі кіберзагроз, та сформулювати практичні рекомендації для організацій із різним рівнем ресурсів.

Розв’язання цих завдань дозволить подолати наявні обмеження у сфері виявлення загроз, підвищити адаптивність систем кіберзахисту та забезпечити науково-практичний внесок у розвиток методів інтеграції штучного інтелекту в інформаційну безпеку.

Висновки до першого розділу

У рамках даного розділу було здійснено комплексний аналіз предметної області, який дозволив визначити науково-практичні передумови для подальшого дослідження методів застосування штучного інтелекту в інформаційній безпеці. Розглянуто сучасні підходи до побудови інтелектуальних систем, зокрема архітектури навчання з учителем, без учителя та з підкріпленням, що дало змогу окреслити переваги й обмеження кожного з них у контексті кіберзахисту. Це дозволило виявити потенціал використання алгоритмів машинного навчання для аналізу великих обсягів даних, адаптації до нових загроз і зниження кількості хибних спрацювань.

Особливу увагу приділено дослідженню основних типів загроз і вразливостей інформаційних систем. Описано характерні механізми атак, серед яких DDoS, фішинг та компрометація ланцюга постачання, проаналізовано методи зловживання штучним інтелектом, зокрема генеративний фішинг, deepfake-соціальну інженерію, автоматизований пошук вразливостей та AI-кероване шкідливе ПЗ. Вивчено внутрішні загрози, включаючи дії зловмисних інсайдерів, помилки користувачів, зловживання адміністративними привілеями та ризики від

третіх сторін, а також фізичні та природні фактори, що становлять суттєвий ризик для стійкості інформаційної інфраструктури. Такий аналіз дозволив сформувати цілісне уявлення про багатовимірний характер сучасних кіберзагроз.

Проведено аналіз існуючих інструментів і програмних рішень для автоматизованого виявлення атак. Вивчено архітектуру та функціональні можливості Snort, Suricata та Zeek, визначено їхні переваги й обмеження, а також роль у сучасній екосистемі засобів кіберзахисту, що включає IDS/IPS, SIEM, EDR/EPP, XDR та платформи SOAR. Це дало змогу окреслити, які завдання ефективно розв’язують наявні рішення, а які аспекти залишаються відкритими і потребують застосування інтелектуальних методів.

На основі проведеного аналізу було сформульовано завдання дослідження та постановку проблеми, що полягає у відсутності універсальних і достатньо гнучких підходів до інтеграції методів штучного інтелекту в системи кіберзахисту. Сформовані завдання дозволяють перейти до наступного етапу дослідження, який передбачає розробку методичних основ застосування алгоритмів машинного навчання, побудову математичних моделей і визначення критеріїв оцінки їхньої якості, що буде розглянуто у наступному розділі.

РОЗДІЛ 2. РОЗРОБКА МЕТОДИЧНИХ ОСНОВ ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ В ІНФОРМАЦІЙНІЙ БЕЗПЕЦІ

2.1 Оцінка та порівняння методів машинного навчання для виявлення кібератак

Ефективність систем штучного інтелекту, призначених для виявлення кібератак, безпосередньо залежить від обраних алгоритмів машинного навчання та підходів до аналізу мережевого трафіку. Основним завданням таких систем є здатність точно розпізнавати потенційно шкідливу активність серед великої кількості звичайних з'єднань, мінімізуючи кількість помилкових спрацювань. Для цього використовуються методи класифікації, здатні встановлювати закономірності між числовими характеристиками потоків даних та їхнім статусом – «нормальний» або «атака».

У сучасній практиці інформаційної безпеки широкого поширення набули як класичні алгоритми, що включають логістичну регресію, дерева рішень і методи ансамблевого навчання, так і просунуті моделі, побудовані на нейронних мережах і глибоких архітектурах [32] . Кожен із цих підходів має власні переваги: лінійні моделі забезпечують високу інтерпретованість результатів, тоді як ансамблеві алгоритми, зокрема «швидке дерево», здатні досягати вищої точності класифікації за рахунок об'єднання багатьох слабких моделей у єдиний прогнозуючий механізм.

Особливістю задачі виявлення кібератак є потреба у реальному або наближеному до реального часу аналізі даних, що вимагає оптимального співвідношення між швидкодією моделі та її аналітичною здатністю. Методи машинного навчання, орієнтовані на обробку числових і категоріальних ознак (таких як тривалість з'єднання, обсяг переданих байтів, частота пакетів, тип протоколу тощо), дозволяють сформувати узагальнену модель поведінки мережі та виявляти відхилення, характерні для вторгнень або аномальних дій користувачів.

Метод *градієнтного бустингу* (англ. Gradient Boosting) належить до класу ансамблевих алгоритмів машинного навчання, які поєднують результати кількох слабких моделей (переважно дерев рішень) для формування одного потужного

прогнозуючого класифікатора [29]. Його ключова ідея полягає у покроковому вдосконаленні моделі, де кожне наступне дерево намагається виправити помилки попередніх, мінімізуючи різницю між реальними значеннями та передбаченими результатами. Таким чином, модель поступово «навчається на власних помилках», підвищуючи точність прогнозів.

Градієнтний бустинг широко застосовується в задачах класифікації, регресії, ранжування та виявлення аномалій. Він ефективний у випадках, коли дані мають велику кількість змішаних ознак (числових, категоріальних), а залежності між ними є нелінійними [30]. Завдяки цій гнучкості метод використовується у фінансових системах для виявлення шахрайських транзакцій, у промисловій аналітиці для прогнозування відмов обладнання, а також у системах рекомендаційного типу та медичній діагностиці.

У сфері інформаційної безпеки метод градієнтного бустингу є одним із найрезультативніших для виявлення вторгнень, аномальної поведінки користувачів та ідентифікації шкідливого мережевого трафіку. Завдяки своїй здатності поетапно вдосконалювати прогноз на основі попередніх похибок, цей метод забезпечує високу точність розпізнавання навіть у складних і багатовимірних просторах ознак. Оскільки дані мережевого моніторингу зазвичай містять численні показники – швидкість передавання пакетів, обсяг трафіку, порти з’єднань, типи протоколів, час сесій, кількість активних з’єднань, прапори TCP тощо, – саме ансамблеві моделі дають змогу ефективно інтегрувати всі ці параметри в єдиний аналітичний механізм. Крім того, градієнтний бустинг здатний виявляти приховані нелінійні взаємозв’язки між характеристиками трафіку, що часто залишаються непоміченими при використанні простіших методів класифікації. Це робить його надзвичайно корисним для систем виявлення мережевих аномалій, які потребують не лише швидкого, а й обґрунтованого аналізу поведінкових шаблонів у реальному часі.

На рис. 2.1 подано узагальнену схему роботи методу градієнтного бустингу, що демонструє процес послідовного навчання ансамблю моделей. Зображена послідовність відображає ітеративний характер алгоритму, у якому кожен

наступний етап спрямований на зменшення похибок, допущених попередньою моделлю. Такий підхід забезпечує поступове підвищення точності прогнозу та формування стійкого класифікатора.

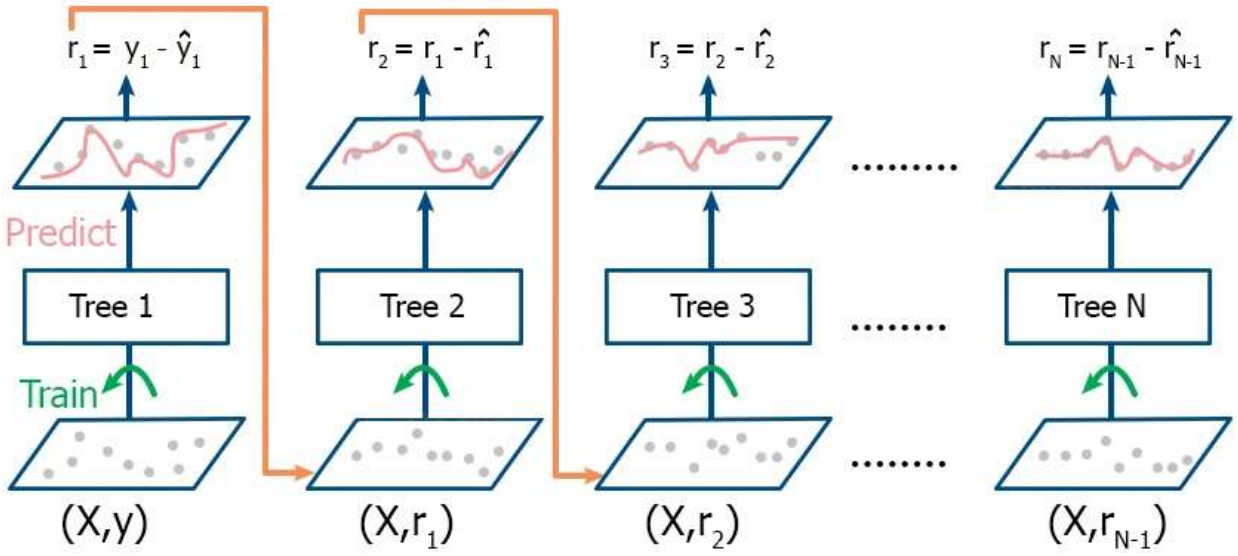

Рис. 2.1 Схема роботи методу градієнтного бустингу [29]

Перший крок методу полягає у створенні базової моделі $h_1(X)$, яка формує початковий прогноз $\hat{y}_1$ на основі навчальної вибірки (X,y). Після цього обчислюється вектор залишків:

$$r_1 = y - \hat{y}_1 \,, \tag{2.1}$$

який характеризує різницю між реальними та передбаченими значеннями цільової змінної. Наступна модель $h_2(X)$ будується вже не на вихідних даних, а на цих залишках, намагаючись змоделювати відхилення попереднього прогнозу.

Кожна наступна ітерація методу градієнтного бустингу спрямована на уточнення результатів попередніх кроків шляхом часткового оновлення прогнозу за формулою:

$$\hat{y}_{i+1} = \hat{y}_i + \eta \cdot h_i(X), \tag{2.2}$$

де $h_i(X)$ – це нова модель, що апроксимує залишкову похибку, а η – коефіцієнт навчання (learning rate), який визначає швидкість оновлення прогнозу та запобігає перенавчанню.

Таким чином, кожна послідовна модель у процесі градієнтного бустингу не створює самостійного прогнозу, а вдосконалює попередній, коригуючи його у

напрямі зменшення градієнта функції втрат. У результаті остаточний прогноз формується як сума внесків усіх проміжних моделей, що дозволяє відтворювати складні нелінійні закономірності між ознаками та досягати високої точності класифікації навіть у багатовимірних просторах ознак.

Переваги методу градієнтного бустингу:

– висока точність прогнозування . Завдяки послідовному вдосконаленню моделі на основі залишкових похибок градієнтний бустинг демонструє одну з найкращих результативностей серед класичних методів машинного навчання, особливо при роботі зі складними та нелінійними залежностями між ознаками;

– універсальність застосування. Метод придатний як для задач класифікації, так і для регресії, ранжування чи виявлення аномалій. Його можна ефективно використовувати у різних галузях – від фінансового аналізу до кібербезпеки – без потреби радикальної зміни архітектури;

– стійкість до різнорідних даних. Градієнтний бустинг добре працює з ознаками різного типу (числовими, категоріальними), не потребує попереднього масштабування даних і здатний враховувати складні взаємозв’язки між атрибутами, що підвищує його адаптивність у реальних умовах.

Недоліки методу градієнтного бустингу:

– висока обчислювальна складність. Через послідовне навчання великої кількості моделей метод потребує значних обчислювальних ресурсів, особливо на великих наборах даних, що ускладнює його використання у системах з обмеженою продуктивністю;

– схильність до перенавчання [33]. Без ретельного налаштування параметрів (кількість ітерацій, глибина моделей, коефіцієнт навчання) градієнтний бустинг може «підлаштовуватися» під навчальні дані, що призводить до зниження узагальнювальної здатності моделі;

– складність інтерпретації. На відміну від лінійних моделей, ансамблі бустингу мають складну структуру, тому пояснити вплив кожної ознаки на кінцеве рішення важко. Це може бути проблемою у сферах, де потрібна прозорість рішень, наприклад, у фінансовому або медичному аналізі.

Отже, метод градієнтного бустингу є потужним інструментом побудови високоточних моделей машинного навчання, здатних адаптуватися до складних і неоднорідних даних. Його ефективність особливо проявляється у сфері інформаційної безпеки, де точність і надійність класифікації мають критичне значення для своєчасного виявлення кібератак і аномальної активності. Попри підвищені вимоги до ресурсів і складність налаштування, правильно оптимізований градієнтний бустинг забезпечує оптимальний баланс між точністю, гнучкістю та стійкістю, що робить його одним із провідних методів у сучасних аналітичних системах кіберзахисту.

Метод *швидкого дерева* (англ. FastTree, ШД) належить до високопродуктивних алгоритмів машинного навчання, призначених для вирішення задач класифікації, регресії та ранжування [34]. Його основна ідея полягає у послідовному побудуванні ансамблю простих моделей, які спільно формують підсумкове рішення. Завдяки спеціальним оптимізаційним механізмам ШД забезпечує високу точність прогнозування за відносно короткий час навчання, що робить його придатним для практичного використання в аналітичних системах.

ШД створює модель, що поступово покращує результати попередніх кроків, фокусуючись на тих прикладах, де попередні передбачення були неточними. Такий підхід дозволяє моделі з кожною ітерацією підвищувати точність і зменшувати загальну похибку. На виході формується ансамбль прогнозів, об’єднаних у єдину узгоджену систему рішень, яка може ефективно відтворювати складні взаємозв’язки між вхідними ознаками.

У галузі інформаційної безпеки метод ШД демонструє високу ефективність під час аналізу та класифікації мережевого трафіку, виявлення аномалій, несанкціонованих дій або підозрілої активності [35]. Оскільки вхідні дані в таких системах зазвичай складаються з великої кількості параметрів – швидкість передавання пакетів, тривалість з’єднання, кількість переданих байтів, тип протоколу, порт джерела тощо, – ШД дозволяє створювати моделі, здатні визначати характер трафіку з високою точністю.

На рис. 2.2 зображено послідовність роботи методу швидкого дерева, який оперує кількома слабкими класифікаторами (окремими деревами рішень), кожне з яких навчається на своїй версії даних. Спочатку алгоритм аналізує вихідний набір даних і створює першу модель, що генерує базовий прогноз. Далі система порівнює цей прогноз із реальними результатами та виявляє помилки класифікації. На наступних етапах ці помилки враховуються – дані, які були класифіковані неправильно, отримують більшу вагу, завдяки чому наступна модель зосереджується саме на складних для розпізнавання випадках.

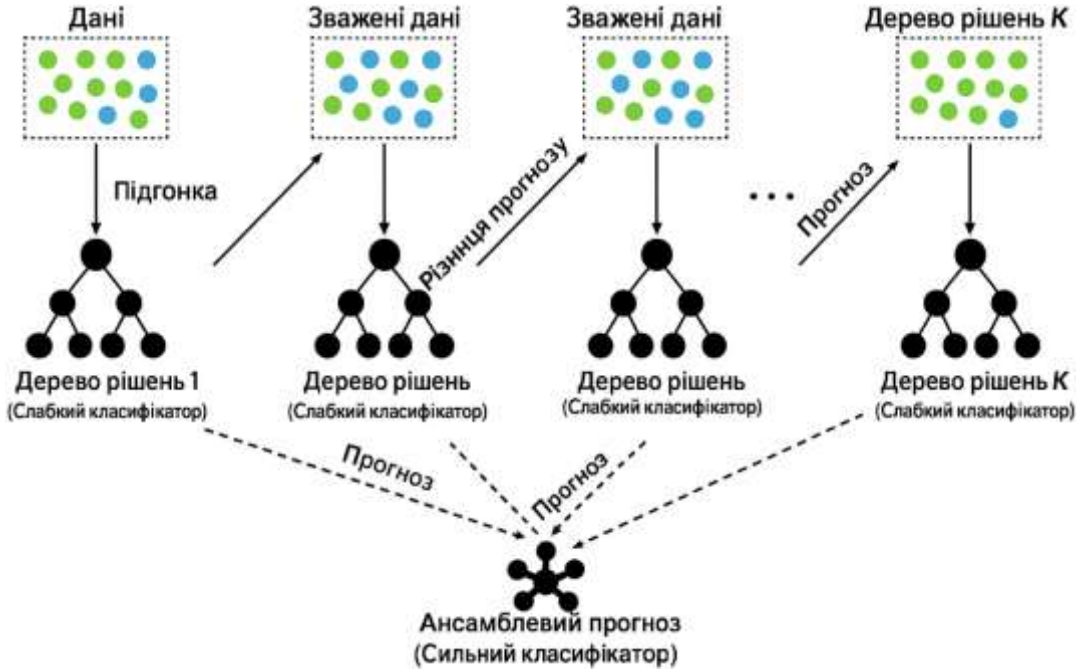

Рис. 2.2 Схема роботи методу швидкого дерева [36]

Кожне дерево виконує власне часткове передбачення, яке саме по собі може бути неточним, однак у сукупності їхні результати утворюють ансамблевий прогноз. Цей прогноз є зваженим об’єднанням рішень усіх моделей, що дозволяє компенсувати індивідуальні похибки кожного класифікатора та сформувати сильний, узагальнюючий результат. Метод ШД поєднує у собі велику кількість простих моделей, які колективно забезпечують високу точність класифікації навіть у складних багатовимірних задачах.

Переваги методу швидкого дерева:

– висока швидкодія та масштабованість [37]. Оптимізований для роботи з великими наборами даних, завдяки чому забезпечує швидке навчання і

прогнозування навіть у системах із обмеженими обчислювальними ресурсами. Це робить його придатним для інтеграції в реальні аналітичні рішення, що працюють у режимі близькому до реального часу;

– стійкість до складних структур даних. Ефективно працює з числовими та категоріальними ознаками, враховує нелінійні залежності між параметрами та демонструє стабільні результати навіть за наявності шуму або частково відсутніх даних;

– добре узагальнення результатів. За рахунок ансамблевої природи метод поєднує рішення кількох моделей, що дозволяє зменшити ризик локальних помилок і досягати високої точності класифікації. У контексті інформаційної безпеки це забезпечує надійне виявлення аномалій і підозрілих дій.

Недоліки методу швидкого дерева:

– складність інтерпретації результатів. Як і більшість ансамблевих методів, формує прогноз на основі великої кількості внутрішніх рішень, тому зрозуміти, які саме фактори визначили кінцевий результат, буває непросто. Це ускладнює пояснення поведінки моделі в критичних системах безпеки;

– потреба в налаштуванні гіперпараметрів. Для досягнення оптимальної точності необхідно ретельно підібрати параметри, такі як кількість ітерацій, глибину моделі, розмір кроку навчання тощо. Невдало підібрані параметри можуть призвести до зниження точності або перенавчання;

– зростання вимог до пам’яті з розміром ансамблю [38]. Із кожною новою ітерацією кількість проміжних моделей збільшується, що потребує додаткових ресурсів для зберігання та обчислення. Це може бути обмежувальним фактором при роботі з дуже великими наборами даних.

Отже, метод ШД є потужним та ефективним інструментом для побудови аналітичних моделей, що поєднує високу точність, швидкість обробки та гнучкість у роботі з великими обсягами даних. Його використання в інформаційній безпеці дозволяє реалізовувати системи, здатні своєчасно розпізнавати аномалії, фіксувати потенційні кібератаки й адаптуватися до змін у поведінці мережевого трафіку. Незважаючи на певну складність налаштування та інтерпретації, ШД залишається

одним із найефективніших методів для створення інтелектуальних систем виявлення загроз і підтримки процесів кіберзахисту.

Логістична регресія – це один із базових і водночас найважливіших методів машинного навчання, який застосовується для розв'язання задач бінарної класифікації, тобто визначення, до якого з двох можливих класів належить спостереження [39]. Незважаючи на свою назву, цей метод не виконує регресію в класичному розумінні, а будує модель, що прогнозує ймовірність належності об'єкта до певного класу.

У сфері інформаційної безпеки логістична регресія використовується для ідентифікації, класифікації та прогнозування кіберзагроз, оскільки вона здатна точно моделювати ймовірність виникнення інциденту безпеки. Типові напрямки застосування включають [40]:

– виявлення шкідливого трафіку. Модель може аналізувати мережеві пакети за такими параметрами, як розмір, частота, напрямок і тривалість з'єднання, прогнозуючи, чи є трафік потенційно небезпечним;

– аналіз поведінки користувачів. Використання поведінкових метрик (кількість входів у систему, частота доступу до файлів, час активності тощо) дозволяє виявляти аномальні дії, що можуть свідчити про компрометацію облікового запису;

– визначення ймовірності фішингових атак або спаму. На основі текстових, структурних і технічних ознак повідомлень логістична регресія може ефективно класифікувати вхідні електронні листи чи веб-домени як безпечні або підозрілі.

– оцінка ризику доступу. Модель може прогнозувати, чи є певна спроба автентифікації легітимною, базуючись на історії дій користувача, IP-адресі, пристрої та геолокації.

На рис. 2.3 подано узагальнену схему роботи методу логістичної регресії, що відображає послідовність основних етапів оброблення даних – від надходження вхідних параметрів до формування кінцевого прогнозу.

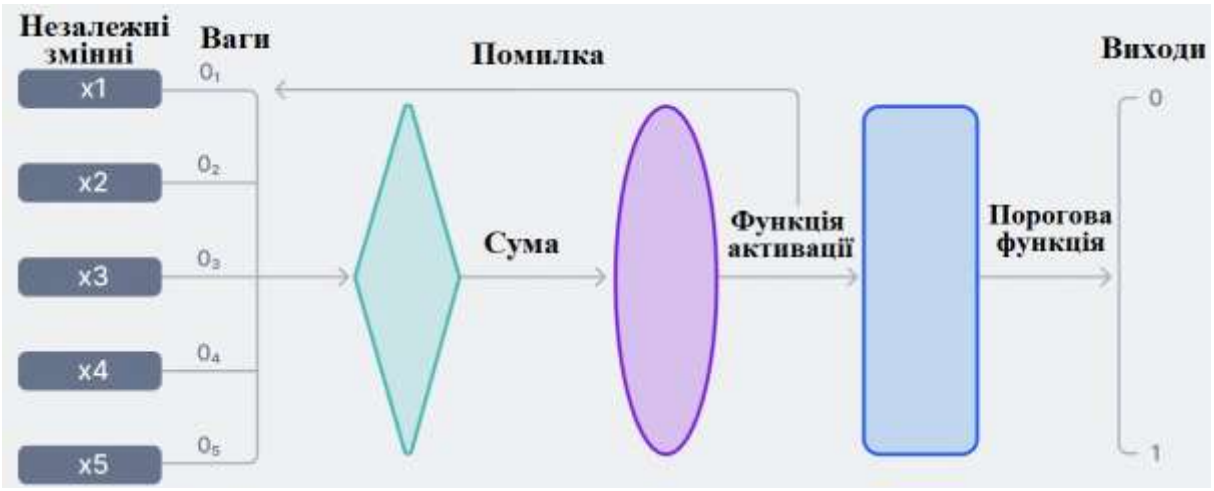

Рис. 2.3 Схема роботи методу логістичної регресії [41]

На рис. 2.3 видно, що вхідні дані представлені у вигляді незалежних змінних x_1, x_2, …, x_n, які характеризують певний об'єкт або подію (наприклад, параметри мережевого з'єднання, поведінкові ознаки користувача чи властивості пакета даних). Кожна змінна має свій ваговий коефіцієнт ω_i, який визначає її вплив на підсумковий результат. На першому етапі модель обчислює зважену суму всіх входів:

$$z = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n, \tag{2.3}$$

де w_0 є вільним коефіцієнтом (зміщенням).

Отримане значення z передається через функцію активації, роль якої у логістичній регресії виконує сигмоїдна функція:

$$\sigma(z) = \frac{1}{1+e^{-z}}. \tag{2.4}$$

Ця функція перетворює результат у діапазон від 0 до 1, що дозволяє інтерпретувати його як ймовірність належності об'єкта до позитивного класу. Наприклад, якщо $\sigma(z)=0,8$, це означає, що модель оцінює ймовірність події «атака» на рівні 80 %.

Далі результат проходить через порогову функцію, яка приймає кінцеве рішення: якщо отримане значення перевищує визначений поріг (зазвичай 0,5), об'єкт відноситься до класу 1 (наприклад, «виявлено загрозу»), інакше – до класу 0 («загроза відсутня»).

Зворотний зв’язок у вигляді помилки використовується для коригування вагових коефіцієнтів під час навчання моделі. Чим більша різниця між прогнозом та фактичним результатом, тим сильніше коригуються ваги відповідних ознак, щоб зменшити похибку в подальших ітераціях.

Переваги методу логістичної регресії:

– інтерпретованість і прозорість результатів [42]. На відміну від складніших моделей, логістична регресія дозволяє чітко зрозуміти, як кожна вхідна змінна впливає на кінцевий результат. Коефіцієнти моделі мають безпосереднє тлумачення – вони показують напрямок і силу впливу параметрів на ймовірність події. Це особливо важливо у сферах, де потрібна пояснюваність рішень, наприклад, у кіберзахисті або фінансовому аудиті;

– висока швидкість навчання і невеликі обчислювальні витрати. Метод добре підходить для великих наборів даних і може швидко оновлюватися при надходженні нової інформації. Це робить логістичну регресію придатною для систем моніторингу безпеки, що працюють у режимі реального часу;

– стійкість і узагальненість результатів. Логістична регресія демонструє стабільну поведінку навіть при невеликій кількості ознак і не потребує складних гіперпараметрів. Вона здатна ефективно працювати на збалансованих даних і забезпечує надійні прогнози у завданнях із двома класами (наприклад, «загроза/безпека»).

Недоліки методу логістичної регресії:

– обмеженість у роботі з нелінійними залежностями. Логістична регресія моделює лінійний зв’язок між ознаками та логарифмом відношення ймовірностей, тому складні взаємодії між параметрами вона відображає недостатньо точно. Це обмежує її ефективність у високорозмірних або гетерогенних даних кібербезпеки;

– вимогливість до якості даних [43]. Модель чутлива до викидів, мультиколінеарності та відсутніх значень, тому перед навчанням необхідна ретельна підготовка – нормалізація, очищення та відбір релевантних ознак;

– низька гнучкість при великій кількості параметрів. За наявності великої кількості ознак або складних нелінійних взаємозв’язків точність моделі може різко

знижуватись, що потребує додаткових методів розширення (наприклад, регуляризації або поліноміальних перетворень).

Отже, метод логістичної регресії залишається одним із найбільш збалансованих і надійних інструментів для побудови моделей класифікації, особливо коли важлива інтерпретованість результатів. Його застосування в інформаційній безпеці дозволяє точно оцінювати ймовірність загроз, виявляти аномалії в поведінці користувачів і визначати потенційно шкідливу активність. Незважаючи на обмеження у відображенні складних нелінійних процесів, логістична регресія є ефективною відправною точкою для створення аналітичних систем кіберзахисту, які поєднують швидкість, простоту та надійність прийняття рішень.

2.2 Вибір та обґрунтування використаного набору даних для навчання та тестування моделей

Для навчання та тестування моделей, призначених для виявлення кіберзагроз, у даному дослідженні було обрано набір даних CyberFedDefender, розміщений на платформі Kaggle [44]. Цей датасет створено спеціально для досліджень у сфері інтелектуального виявлення кіберзагроз, виявлення аномалій та розподілених систем кіберзахисту. Його структура та зміст орієнтовані на сучасні умови оброблення мережевого трафіку, зокрема у хмарних і периферійних (edge) середовищах, де питання приватності, швидкодії та точності моделей машинного навчання є критично важливими.

Вибір цього набору даних зумовлений декількома ключовими факторами, зокрема:

– CyberFedDefender містить реалістично змодельовані мережеві сесії, що охоплюють як звичайний (нормальний) трафік, так і типові види атак, зокрема DDoS, Brute Force та Ransomware. Це дозволяє досліджувати роботу моделей у різних сценаріях кіберзагроз;

– датасет характеризується збалансованою структурою та наявністю 23 ключових атрибутів, які відображають технічні властивості мережевого трафіку,

такі як розмір пакетів, кількість переданих байтів, швидкість потоку, типи протоколів і порти з’єднання. Така детальна структура забезпечує можливість комплексного аналізу поведінки мережевих процесів, що є фундаментом для побудови ефективних моделей класифікації;

– набір даних містить мітки атак і нормального трафіку, що робить його повністю придатним для задач навчання з учителем. Завдяки цьому його можна використовувати для побудови, навчання й оцінювання моделей машинного навчання у системах виявлення вторгнень, аналізу поведінкових патернів користувачів, а також у розподілених сценаріях кіберзахисту, де важливим є збереження конфіденційності даних;

– CyberFedDefender оптимізований для експериментів у контексті federated learning – підходу, що дозволяє тренувати моделі без прямого доступу до всіх даних, що є актуальним для корпоративних та хмарних систем безпеки, де дані розподілені між різними вузлами мережі.

Обраний набір даних повністю відповідає меті дослідження – розробці та оцінюванню моделей машинного навчання для виявлення кіберзагроз на основі поведінкового аналізу трафіку. У табл. 2.1 наведено основні атрибути датасету, які використовуються для побудови моделей класифікації мережевих з’єднань.

Таблиця 2.1

Атрибути набору даних CyberFedDefender

№	Назва атрибута	Опис
1	2	3
1	Timestamp	Час фіксації мережевого трафіку, що дозволяє відстежити послідовність подій
2	Source_IP	IP-адреса пристрою, з якого надходить трафік
3	Destination_IP	IP-адреса отримувача пакета
4	Protocol	Тип мережевого протоколу (TCP, UDP, ICMP), що використовується для з’єднання
5	Packet_Length	Довжина мережевого пакета в байтах

Продовження таблиці 2.1.

1	2	3
6	Duration	Тривалість сесії або з’єднання в секундах
7	Source_Port	Номер порту джерела з’єднання
8	Destination_Port	Номер порту призначення
9	Bytes_Sent	Загальна кількість байтів, переданих від джерела до отримувача
10	Bytes_Received	Загальна кількість байтів, отриманих відправником у відповідь
11	Flags	Індикатори стану TCP-з’єднання (наприклад, SYN, ACK, FIN)
12	Flow_Packets/s	Кількість пакетів у потоці за секунду
13	Flow_Bytes/s	Кількість байтів, переданих у потоці за секунду
14	Avg_Packet_Size	Середній розмір пакета протягом сесії
15	Total_Fwd_Packets	Загальна кількість пакетів, відправлених у прямому напрямку
16	Total_Bwd_Packets	Загальна кількість пакетів, отриманих у зворотному напрямку
17	Fwd_Header_Length	Довжина заголовків пакетів у прямому потоці
18	Bwd_Header_Length	Довжина заголовків пакетів у зворотному потоці
19	Sub_Flow_Fwd_Bytes	Кількість байтів, переданих у підпотоці в прямому напрямку
20	Sub_Flow_Bwd_Bytes	Кількість байтів, отриманих у підпотоці у зворотному напрямку
21	Inbound	Ознака напрямку трафіку: 1 – вхідний, 0 – вихідний
22	Attack_Type	Тип атаки або позначка “Normal” для звичайного трафіку
23	Label	Бінарна мітка класу: 1 – шкідливий трафік, 0 – нормальний

Для забезпечення коректності побудови моделей машинного навчання було проведено попередній аналіз обраного набору даних CyberFedDefender, який охоплював дослідження структури, повноти, типів ознак і наявності можливих кореляцій між ними. Такий аналіз дозволяє виявити надлишкові або тісно пов’язані між собою змінні, що можуть впливати на стабільність і точність моделі. Крім того, він допомагає визначити ступінь взаємозалежності між технічними характеристиками мережевого трафіку, що є критично важливим у задачах кібербезпеки, де навіть незначні кореляційні зв’язки можуть мати діагностичне значення.

На рис. 2.4 представлено теплову карту, що демонструє кореляційні залежності між кількісними змінними набору даних. Вісь X і вісь Y містять

однаковий набір атрибутів, а кольорова шкала справа ілюструє силу зв’язку: від синього кольору (негативна кореляція) до червоного (позитивна кореляція). Як видно з рисунка, більшість клітинок мають світлі відтінки, що свідчить про слабкий або помірний рівень кореляції між ознаками. Це означає, що вибрані параметри трафіку є відносно незалежними, що є позитивним фактором для навчання моделей, оскільки зменшується ризик мультиколінеарності.

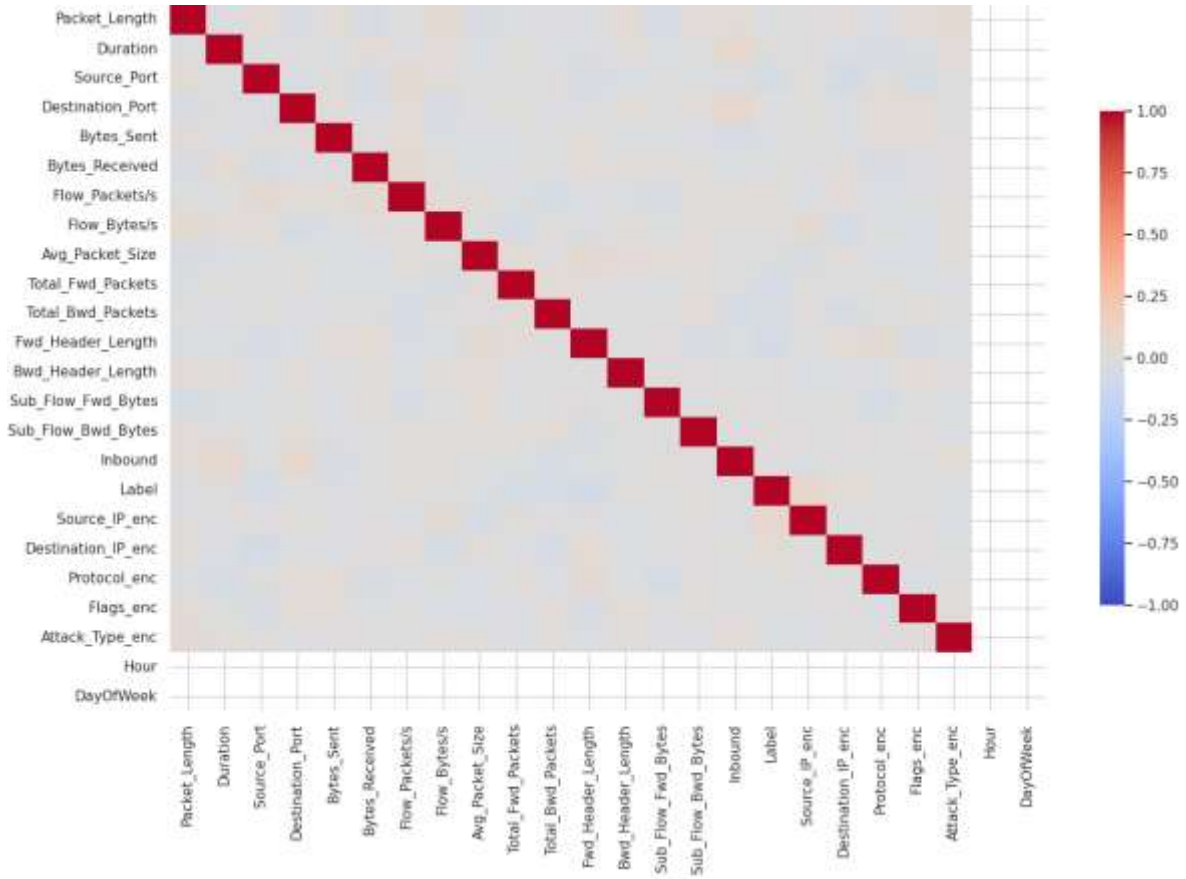

Рис. 2.4 Теплова карта кореляції

Серед помірно корельованих пар можна відзначити ознаки, що описують обсяги та швидкості потоків – наприклад, Bytes_Sent, Bytes_Received, Flow_Bytes/s і Avg_Packet_Size, які мають логічно обґрунтований взаємозв’язок, адже характеризують інтенсивність передавання даних у межах однієї сесії. Водночас такі атрибути, як Source_Port, Destination_Port, Flags або Inbound, не демонструють помітної залежності від інших, що свідчить про їхню унікальну інформативність для класифікації типів трафіку.

Загалом теплова карта підтверджує, що набір даних є структурно збалансованим і придатним для машинного аналізу, оскільки не містить значної

кількості взаємопов’язаних ознак, які могли б спотворювати результати моделі. Такий рівень кореляційної незалежності створює передумови для ефективного навчання алгоритмів виявлення кіберзагроз і дозволяє досягати більшої стабільності прогнозів у процесі тестування.

Рис. 2.5 відображає, що протоколи демонструють різний рівень активності у часовому проміжку спостереження. Найбільшу частку трафіку формує UDP, що характерно для потокових сервісів і передачі мультимедійних даних, тоді як TCP займає проміжну позицію, забезпечуючи стабільний обсяг з’єднань, властивий більшості веб-додатків та служб доступу. Протокол ICMP представлений найменшими обсягами даних, проте його активність має важливе діагностичне значення – коливання цього показника можуть свідчити про тестування мережі або початок DDoS-атаки.

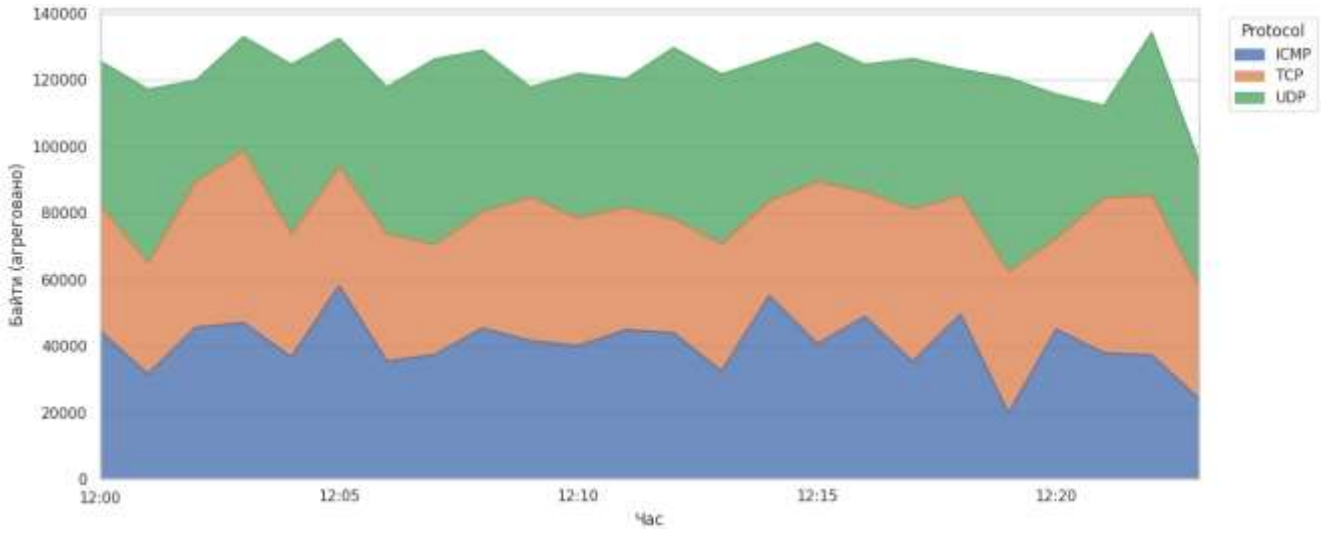

Рис. 2.5 Динаміка трафіку зі стекуванням за протоколами

Графік засвідчує відносну стабільність загального навантаження, однак при цьому спостерігаються незначні піки активності, зумовлені короткочасними змінами інтенсивності передавання пакетів. Така структура трафіку є типовою для середовищ, у яких одночасно функціонує кілька типів сервісів і протоколів. Розподіл байтів між протоколами має багаторівневу природу, а співвідношення ICMP, TCP і UDP може виступати показником нормальної чи аномальної поведінки мережевої системи.

На рис. 2.6 видно, що спостереження розміщені нерівномірно: більшість точок зосереджена у діапазоні середніх значень обсягів переданих і отриманих байтів (до 1500 одиниць), тоді як крайні значення характеризуються меншою щільністю, що може свідчити про рідкісні або нетипові сесії. Колірна шкала праворуч позначає значення змінної Label, яка відображає клас трафіку – нормальний (0) або шкідливий (1).

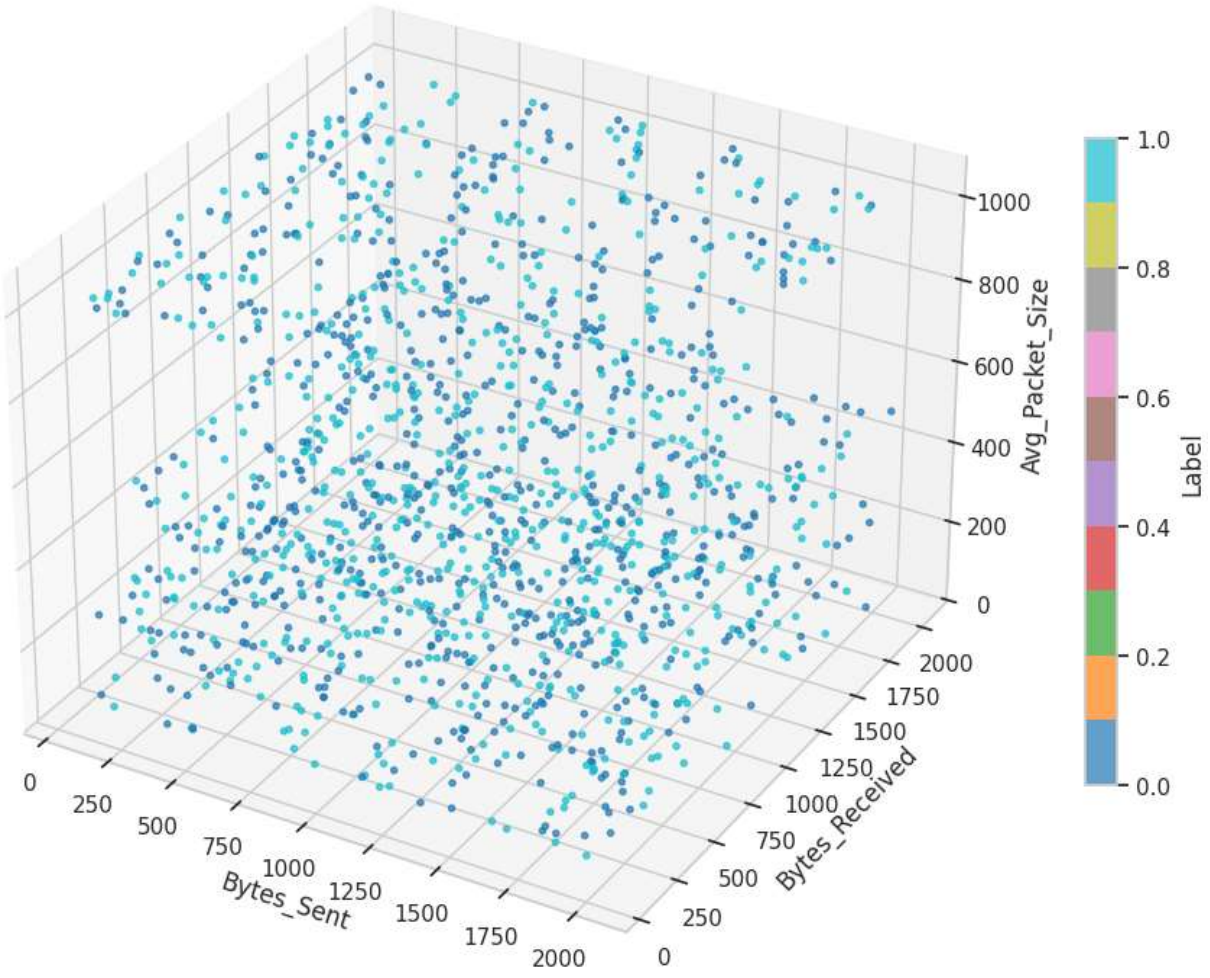

Рис. 2.6 3D-скатер за трьома конкретними ознаками

Розподіл точок свідчить про наявність певної закономірності між середнім розміром пакета та інтенсивністю обміну даними: більші значення параметра Avg_Packet_Size частіше асоціюються з високими обсягами Bytes_Sent і Bytes_Received, що властиво для атак типу DDoS або Brute Force, коли спостерігається підвищена частота передачі пакетів. Натомість спостереження з невеликими обсягами байтів і середнім розміром пакета частіше відповідають звичайній (нормальній) активності.

Загалом тривимірна діаграма дозволяє візуально оцінити структуру простору ознак та потенційну можливість розділення класів, що є важливим перед навчанням моделі класифікації. Такий підхід підтверджує наявність відмінностей у поведінці мережевих потоків різних типів, що створює сприятливі умови для побудови ефективних алгоритмів виявлення кіберзагроз.

На рис. 2.7 шестикутні комірки відображають щільність появи пар значень, де темніші відтінки вказують на більшу кількість спостережень у відповідному діапазоні. Видно, що більшість комірок з високою щільністю розташована в області середніх значень – приблизно між 500 і 1500 байт як для переданих (Bytes_Sent), так і для отриманих (Bytes_Received) даних. Це свідчить про переважання коротких і середньотривалих сесій, характерних для звичайного мережевого трафіку.

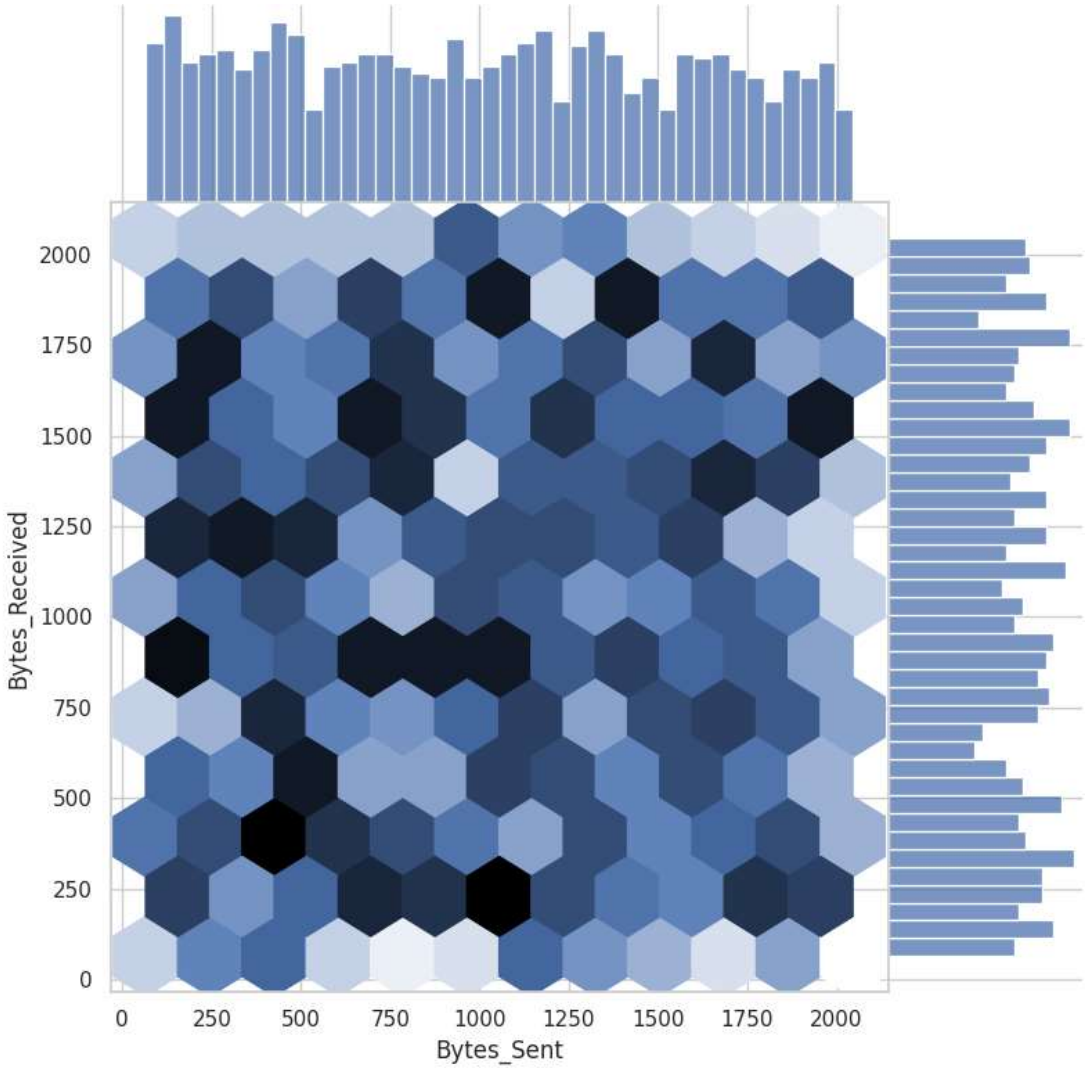

Рис. 2.7 2D-щільність для змінних Bytes_Sent та Bytes_Received

Натомість поодинокі темні осередки у крайніх діапазонах (особливо при великих обсягах переданих даних) можуть вказувати на аномальні активності, наприклад, на спроби масової передачі даних або інтенсивні запити, властиві атакам типу Brute Force чи DDoS. Аналіз такої двовимірної структури дозволяє виділити потенційні зони ризику для подальшої обробки методами машинного навчання.

Рис. 2.8 висвітлює графічне відображення підвибірки числових ознак датасету, спроєктованих у дві координати – t-SNE 1 та t-SNE 2. Кожна точка відповідає окремому мережевому з'єднанню, а колір позначає класову мітку (Label), тобто належність трафіку до нормального або шкідливого типу.

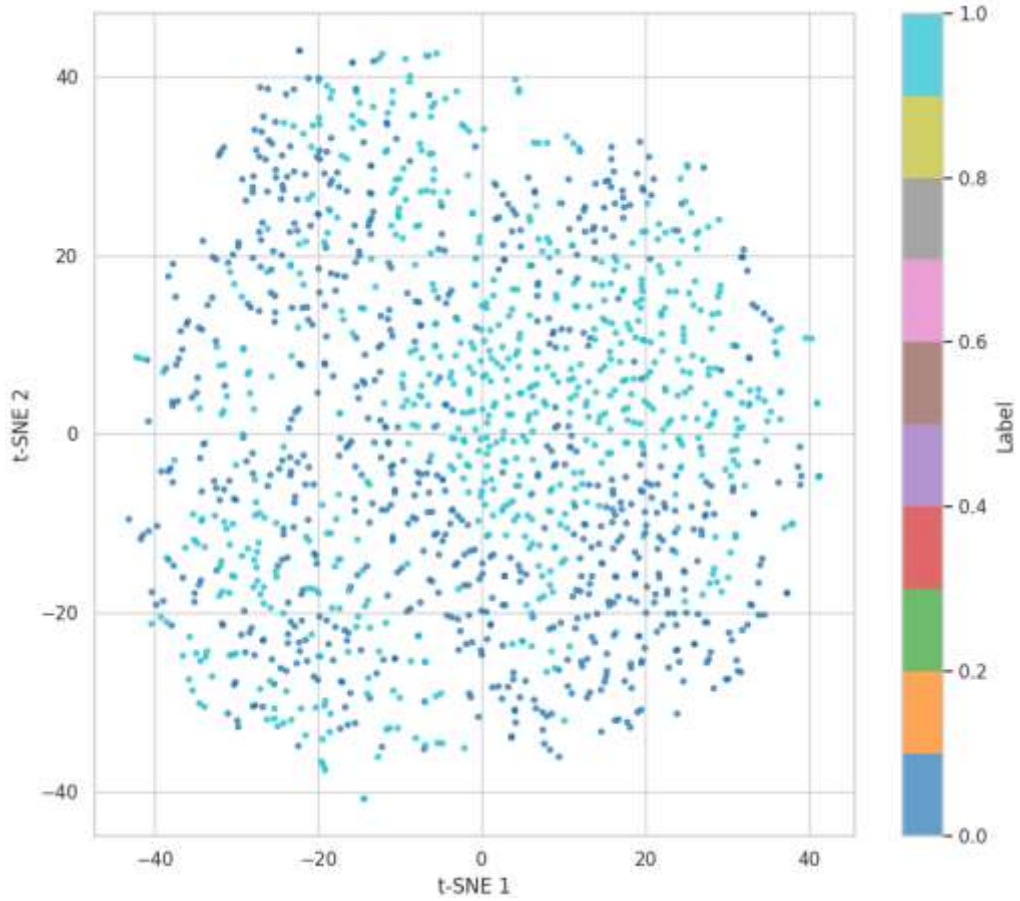

Рис. 2.8 t-SNE 2D на підвибірці числових ознак

Як видно з рисунку, точки розташовані нерівномірно, утворюючи окремі щільні області, що вказує на наявність певної структури в даних. Це підтверджує, що між різними типами трафіку існують закономірні відмінності, які можуть бути виявлені алгоритмами машинного навчання. Зокрема, деякі скупчення точок із

подібними кольорами демонструють групування записів, що належать до однакових типів атак, таких як DDoS чи Brute Force, тоді як інші групи з різними кольорами відображають перетин нормального та шкідливого трафіку.

Отримана візуалізація засвідчує, що t-SNE ефективно відображає приховані взаємозв'язки між ознаками навіть у складних, багатовимірних наборах даних. Такий підхід допомагає досліднику попередньо оцінити придатність вибраних характеристик для подальшої класифікації та визначити потенційні зони перекриття класів, що можуть ускладнювати розмежування нормальної й аномальної поведінки у системах кіберзахисту.

На рис. 2.9 представлено візуалізацію у вигляді паралельних координат, яка дає змогу оцінити взаємозв'язки між кількома числовими ознаками датасету та порівняти профілі різних типів мережевого трафіку. Кожна лінія на графіку відповідає окремому запису, а колір визначає тип активності – нормальний або з ознаками кіберзагрози (DDoS, Ransomware, Brute Force).

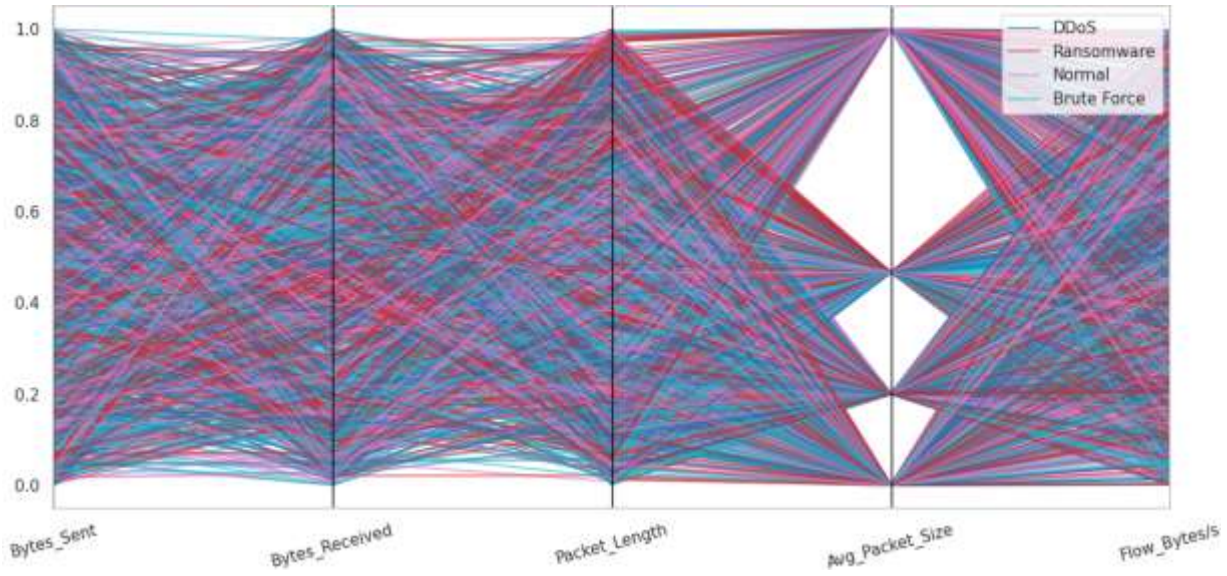

Рис. 2.9 Паралельні координати для огляду мультифакторних відмінностей між класами

Аналіз рисунку показує, що класи трафіку демонструють помітні відмінності у розподілах ключових показників, зокрема Bytes_Sent, Packet_Length і Flow_Bytes/s. Для нормального трафіку характерна відносна стабільність цих параметрів, тоді як зловмисна активність відзначається коливаннями з великими амплітудами, що свідчить про аномальну поведінку потоків даних. Наприклад,

DDoS-атаки мають високу інтенсивність передавання байтів при відносно короткій тривалості сесій, а Brute Force характеризується невеликими, але частими обмінами пакетами.

Завдяки такій формі подання даних виявляються закономірності, неочевидні при аналізі окремих змінних, що робить паралельні координати ефективним інструментом попередньої діагностики для систем виявлення вторгнень. Отримані результати підтверджують наявність чітких мультифакторних відмінностей між нормальним та аномальним мережевим трафіком, що є передумовою для подальшого формування високоточних моделей класифікації у сфері інформаційної безпеки.

Рис. 2.10 відображає ранжований перелік найбільш активних джерел мережевого трафіку (Top-15), що дозволяє оперативно виявити вузли з підвищеною активністю та визначити їхній внесок у загальне навантаження мережі. На наведеній діаграмі найбільшу частку записів генерує IP-адреса 192.168.0.1, тоді як інші адреси утворюють поступово спадний хвіст – це вказує на явно виражений «хвіст» активності, коли невелика кількість хостів відповідає за значну частину трафіку.

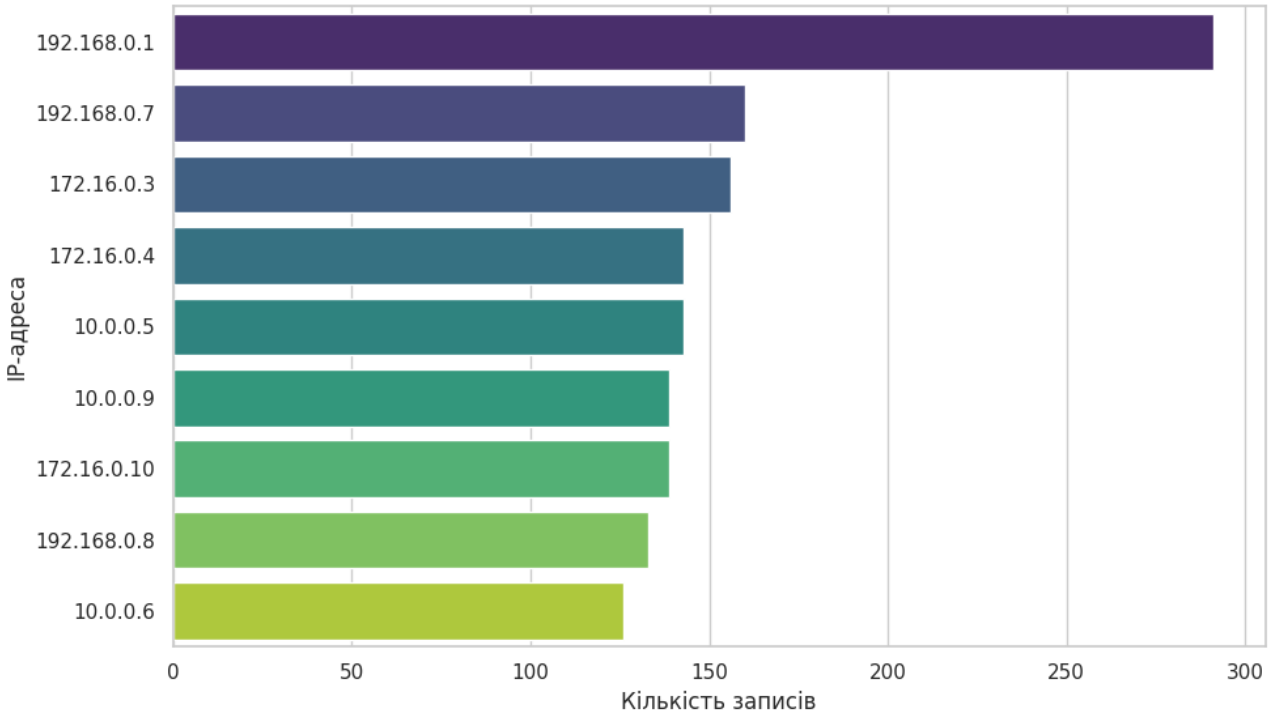

Рис. 2.10 Топ-15 джерел трафіку

Таке скупчення активності може мати як легітимні пояснення (сервери, проксі, централізовані служби), так і свідчити про підозрілу поведінку (сканування, автоматизовані запити, координовані атаки). Тому для оцінки ризику доцільно зіставити ці джерела з мітками атак у наборі даних, проаналізувати часові патерни їх активності та порти/протоколи, що використовуються.

Практичні рекомендації: провести додаткову верифікацію топ-джерел – reverse DNS, перевірку відповідності бізнес-службам, а також застосувати правила фільтрації або обмеження швидкості для аномальних потоків. Також корисно інтегрувати ці результати з модулем кореляції подій у SIEM, щоб автоматично відслідковувати зміни в ранжуванні джерел і вчасно реагувати на потенційні інциденти.

Аналіз розподілу обсягу переданих байтів між різними типами трафіку надає важливі відомості про характер поведінки мережевих потоків у межах досліджуваного набору даних. На рис. 2.11 наведено графік щільності розподілу змінної Bytes_Sent для різних класів трафіку, що дає змогу оцінити статистичні закономірності обсягу переданих даних залежно від типу активності.

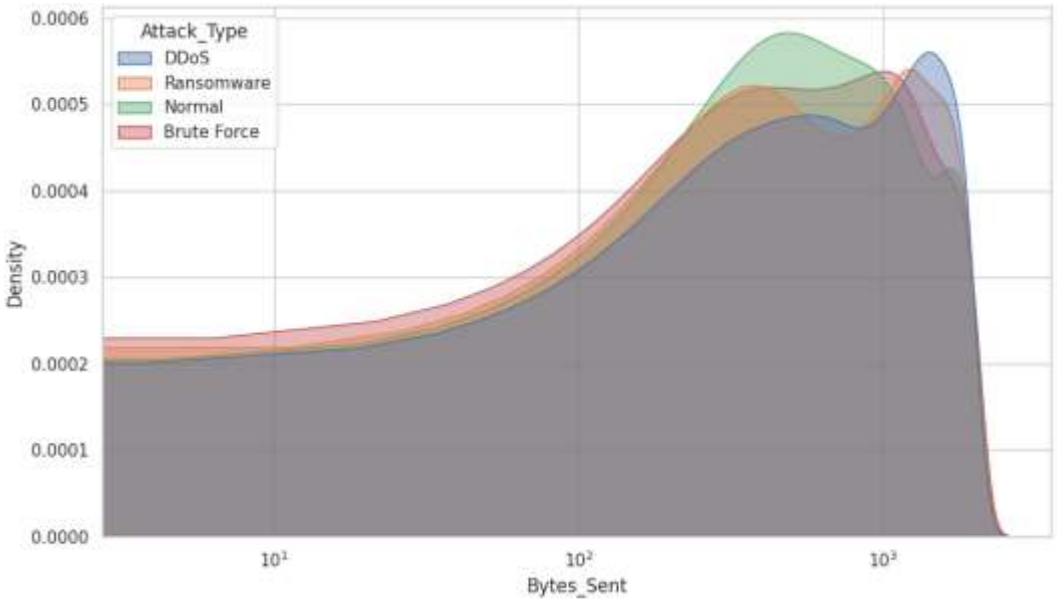

Рис. 2.11 Розподіл трафіку за Label

Як видно з рисунка, нормальний трафік (позначений зеленим кольором) має більш концентрований розподіл у середній області значень, що відповідає звичайній роботі користувачів без надмірних коливань обсягу переданих даних. На

відміну від цього, трафік типу DDoS демонструє виражений пік у діапазоні високих значень байтів, що свідчить про масову передачу великої кількості пакетів у короткі проміжки часу – типову ознаку атак розподіленого відмовлення в обслуговуванні. Для Ransomware спостерігається середній рівень переданих даних із підвищеною варіативністю, що може бути зумовлено фрагментованою передачею шкідливого коду або шифрувальних операцій. Клас Brute Force, у свою чергу, характеризується незначними коливаннями в нижній частині шкали, що пояснюється короткочасними, але частими спробами автентифікації.

Отже, що параметр Bytes_Sent має значний потенціал як предиктор для моделей машинного навчання у задачах виявлення аномалій і класифікації типів атак. Висока відмінність розподілів між класами підкреслює інформативність цієї ознаки для формування ефективних систем кіберзахисту.

Лінійна регресія є базовим інструментом статистичного аналізу, який дозволяє оцінити взаємозв’язки між двома кількісними змінними. У даному випадку вона застосована для перевірки залежності між Bytes_Sent (кількість переданих байтів) та Packet_Length (довжина пакету). На рис. 2.12 наведено результат лінійної апроксимації для різних типів мережевого трафіку, де кожен клас позначено окремим кольором.

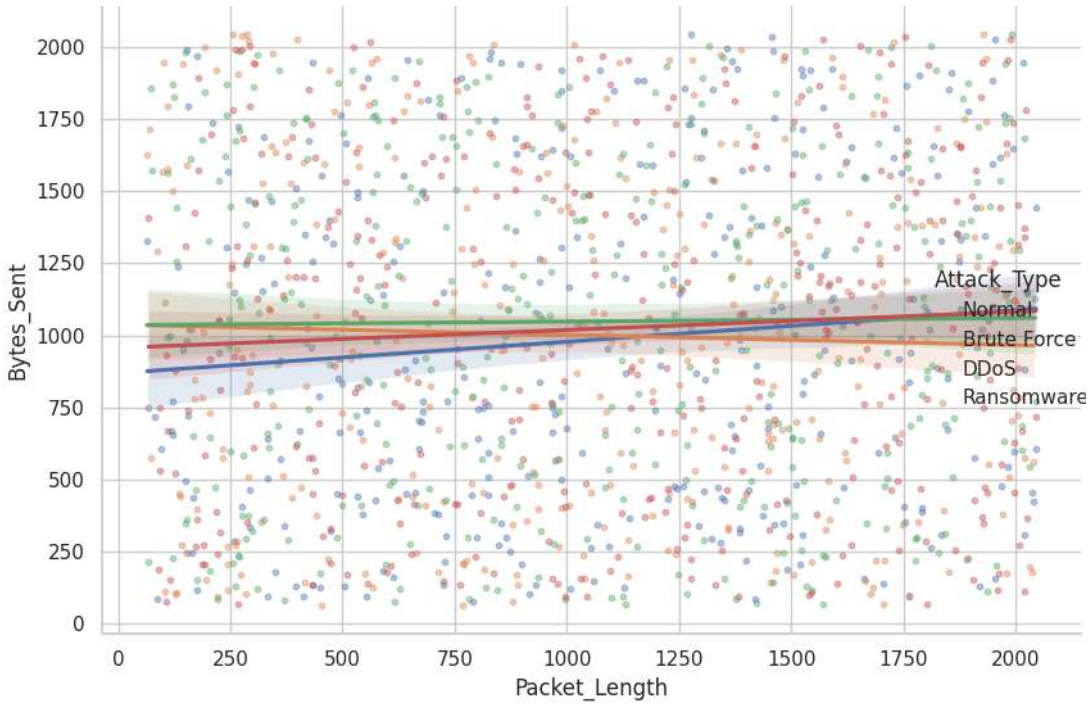

Рис. 2.12 Лінійна регресія між двома змінними Bytes_Sent та Packet_Length

З графіка видно, що для більшості типів трафіку спостерігається помірно додатна кореляція, тобто збільшення довжини пакету загалом супроводжується зростанням обсягу переданих байтів. Найбільш виражений тренд демонструє клас Ransomware, що свідчить про тенденцію використання пакетів великого розміру під час передачі зашифрованих даних. Трафік типу DDoS характеризується незначною, але стабільною тенденцією до збільшення переданих байтів із ростом розміру пакету, тоді як Brute Force та Normal мають більш помірні нахили, що вказує на стабільність передавання невеликих обсягів даних.

Наявність слабкого, але очевидного зв’язку між цими змінними підтверджує, що Packet_Length може бути корисним параметром для класифікаційних моделей машинного навчання, зокрема при формуванні ознакового простору для систем виявлення кібератак. У поєднанні з іншими характеристиками мережевого потоку ця змінна може підвищувати точність і стійкість алгоритмів.

2.3 Побудова математичної моделі для класифікації мережевих загроз

У межах задачі інформаційної безпеки розглядається двокласова класифікація мережевих подій на «атака/норма» з уніфікованим конвеєром оброблення даних; у конвеєрі змінюється лише алгоритм навчання: градієнтний бустинг над деревами рішень, швидке дерево, та логістична регресія. Вибір цих трьох методів зумовлений їхньою комплементарністю: логістична регресія забезпечує інтерпретовану лінійну базу з каліброваними імовірностями; ансамблеві деревні методи моделюють нелінійності та взаємодії ознак без ручної інженерії; ШД добре узгоджений із табличними ознаками та one-hot категоріями, і демонструє високу продуктивність у конвеєрах.

Першим етапом побудови інтелектуальної системи виявлення атак є формалізація навчальної вибірки, що подається як множина спостережень:

$$D = \{x_i, y_i\}_{i=1}^{n}, y_i \in \{0,1\}. \quad (2.5)$$

Тут кожний вектор x_i відображає сукупність характеристик мережевого з'єднання (кількість пакетів, середня довжина, швидкість передавання, порти тощо), а мітка y_i вказує, чи є подія нормальною ($y_i = 0$) або атакою ($y_i = 1$).

Таким чином, задача виявлення вторгнень редукується до бінарної класифікації, у якій необхідно навчити модель $\hat{f}(x)$, що наближає невідому залежність між ознаками та класом події.

У реальних телеметричних даних часто поєднуються числові та категоріальні ознаки. Наприклад, числовими є «кількість байтів», «тривалість з'єднання», а категоріальними – «тип протоколу» чи «TCP-прапорець». Щоб уніфікувати ці різнорідні атрибути для подання в моделі машинного навчання, застосовується one-hot-кодування:

$$\phi OHE := c_i \mapsto u_i \in \{0,1\}^{d_n}. \quad (2.6)$$

Це відображення перетворює кожну категоріальну змінну у вектор нульових і одиничних компонент, що дозволяє уникнути штучного порядку між категоріями та робить простір ознак метричним.

Після кодування категоріальних змінних отримані числові (z_i) та двійкові (u_i) підвектори конкатенуються в єдиний вектор:

$$x_i = [z_i || u_i] \in R^{d_z + d_u}. \quad (2.7)$$

Операція конкатенації формує єдиний простір ознак для подальшої обробки в конвеєрі. Фактично на цьому етапі забезпечується структурна цілісність даних – кожен запис має однакову розмірність і послідовність ознак, що дозволяє застосовувати єдині методи нормалізації та навчання для всіх трьох методів.

Для числових ознак застосовується min-max нормалізація:

$$\bar{x}'_{ij} = \frac{\bar{x}'_{ij} - \min_k \bar{x}'_{kj}}{\max_k \bar{x}'_{kj} - \min_k \bar{x}'_{kj}}, \bar{x}'_{ij} \in [0,1]. \quad (2.8)$$

Її мета – привести всі числові атрибути до єдиного діапазону, щоб уникнути домінування ознак з великими значеннями над іншими. Ця операція особливо важлива для градієнтних методів оптимізації (як-от логістична регресія) та

забезпечує стабільну роботу швидкого дерева, у якому критерії розщеплення залежать від відносних масштабів ознак.

Після нормалізації утворюється вектор $x_i' = \bar{x}_{ij}'$, який слугує стандартизованим входом для моделей.

Для будь-якого з трьох методів навчання – незалежно від того, чи це лінійна логістична регресія, чи ансамбль дерев – результат прогнозування можна інтерпретувати як оцінку умовної ймовірності:

$$p_\theta = (\, y = 1 \mid x \,) \in (0,1), \tag{2.9}$$

де θ – параметри моделі, які визначаються під час навчання.

Таким чином, мета алгоритму полягає у відтворенні невідомої функції розподілу $p(y|x)$, що дозволяє далі приймати рішення за ймовірнісним критерієм.

У контексті інформаційної безпеки така інтерпретація надзвичайно важлива: отримане значення p_θ можна безпосередньо трактувати як ступінь упевненості системи, що аналізована подія є атакою.

Оскільки результатом прогнозу є ймовірність, необхідно визначити правило прийняття рішення – порогову функцію:

$$\hat{y}(x) = I\{p_\theta(y = 1|x) \geq \tau\}, \tau \in (0,1). \tag{2.10}$$

Функція $I\{\cdot\}$ повертає 1, якщо умова істинна, і 0 – в іншому разі. Поріг τ встановлюється з урахуванням співвідношення між ризиком хибного спрацювання та ризиком пропуску атаки. У задачах кіберзахисту часто застосовується занижене значення τ – це дозволяє збільшити показник повноти, навіть якщо зростає кількість хибних тривог.

Після підготовки даних і нормалізації ознак (формули 2.5–2.10) застосовується логістична регресія – один із найдавніших і водночас найінтерпретованіших методів машинного навчання для бінарної класифікації. Її сутність полягає в апроксимації логіт-функції (тобто логарифма відношення шансів події) лінійною комбінацією вхідних змінних:

$$\eta(x) = w^T x' + b, p(x) = \sigma(\eta(x)), \tag{2.11}$$

де w – вектор ваг ознак, b – зміщення ($bias$), а $\sigma(\cdot)$ – сигмоїдна функція.

Отже, модель визначає ймовірність того, що спостереження x належить до класу «атака». Кожен коефіцієнт w_j відображає силу й напрямок впливу *j*-ої ознаки: позитивні ваги збільшують логіт, тобто підвищують ймовірність атаки, а негативні – зменшують її.

Отже, рівняння (2.11) забезпечує прозоре інтерпретування: можна безпосередньо оцінити, які атрибути (наприклад, кількість пакетів, час з'єднання, тип протоколу) найбільше впливають на рішення.

Перетворення $\eta(x) \mapsto p(x)$ здійснюється за допомогою сигмоїдної функції:

$$\sigma(t) = \frac{1}{1-e^{-t}}. \tag{2.12}$$

Це згладжена монотонна функція, що відображає будь-яке дійсне число *t* у інтервал (0,1). Її форма близька до кумулятивної логістичної кривої: при великих позитивних *t* ймовірність наближається до 1 (висока впевненість в атаці), при великих від'ємних – до 0 (нормальна подія).

Сигмоїда також задає безперервний градієнт похибки, що є критичним для алгоритмів оптимізації: на відміну від жорсткого порогового класифікатора, вона дозволяє обчислювати похідні, необхідні для навчання ваг.

Мета навчання логістичної регресії полягає в підборі параметрів *w* і *b*, які мінімізують логістичну функцію втрат (крос-ентропію):

$$L(w,b) = -\sum_{i=1}^{n}[y_i \ln p(x_i) + (1-y_i)\ln(1-p(x_i))] + \frac{\lambda}{2}\|w\|_2^2, \tag{2.13}$$

Перша частина суми – це середнє негативне правдоподібність, що відображає, наскільки передбачені ймовірності близькі до реальних міток уі; чим вона менша, тим краще модель описує дані. Доданок $\frac{\lambda}{2}\|w\|_2^2$ – це L_2-регуляризація, яка обмежує величину ваг і запобігає перенавчанню. Регуляризаційний коефіцієнт $\lambda>0$ встановлює компроміс між точністю апроксимації й узагальнювальною здатністю моделі.

Щоб мінімізувати втрату (2.13), застосовується градієнтний спуск, який послідовно оновлює параметри у напрямку, протилежному градієнту функції втрат:

$$w^{(t+1)} = w^{(t)} - \eta^{(t)} \nabla_w L\left(w^{(t)}, b^{(t)}\right), \quad (2.14)$$

де $\eta^{(t)}$ – швидкість навчання (learning rate), що визначає довжину кроку на кожній ітерації.

Градієнт $\nabla_w L$ обчислюється аналітично й відображає, наскільки зміна ваг впливає на зменшення похибки.

Завдяки опуклості функції втрат (2.13), логістична регресія має єдиний глобальний мінімум, тому незалежно від початкових ваг метод сходиться до оптимального рішення. Цей процес означає, що модель поступово «навчається» зважувати кожну ознаку – наприклад, збільшує вагу для високих значень трафіку або для певного типу протоколу, якщо вони статистично частіше супроводжують зловмисну активність.

На відміну від логістичної регресії, де передбачається лінійна залежність між ознаками та ймовірністю атаки, градієнтний бустинг формує нелінійну композицію простих моделей – дерев рішень. Це виражається адитивною формулою:

$$F_M(x) = F_0 + \sum_{m=1}^{M} \nu \gamma_m h_m(x), \quad (2.15)$$

де:

$h_m(x)$ – m-те дерево рішень, яке моделює залишкові похибки попередніх ітерацій;

M – кількість дерев у ансамблі;

$\nu \in (0,1]$ – коефіцієнт усадки (learning rate), що контролює крок навчання;

γ_m – ваговий коефіцієнт поточного дерева;

F_0 – початкове наближення (зазвичай константа, що відповідає середній лог-odds імовірності).

Таким чином, модель не намагається відразу знайти складну функцію $F_M(x)$, а поступово наближає її як суму простих елементів, кожен з яких покращує точність попереднього. Ця адитивна стратегія дозволяє поступово зменшувати залишкову помилку, не руйнуючи структуру вже вивчених закономірностей.

Подібно до логістичної регресії, результуюча функція ансамблю $F_M(x)$ інтерпретується як логіт-шкала – тобто значення, що відображає силу

приналежності до класу «атака». Перехід до ймовірності здійснюється через сигмоїду:

$$p(x) = \sigma\big(F_M(x)\big) = \frac{1}{1+e^{F_M(x)}}, \quad (2.16)$$

Ця формула робить ансамбль узгодженим із байєсівською інтерпретацією імовірностей і забезпечує спільність із лінійною моделлю: тепер $F_M(x)$ можна розглядати як узагальнений логіт, який моделює не лише лінійні впливи, а й складні взаємодії між ознаками.

Градієнтний бустинг можна розглядати як нелінійне розширення логістичної регресії, де роль вагових коефіцієнтів відіграють деревоподібні підмоделі. Початкове наближення ансамблю. Для початку навчання необхідно визначити початкове значення ансамблю F_0, яке несе базову інформацію про дисбаланс класів у вибірці:

$$F_0 = \ln\frac{\pi}{\pi-1}, \pi = \frac{1}{n}\sum_{i=1}^{n} y_i. \quad (2.16)$$

Тут π – частка позитивних прикладів (атак).

Цей вираз є логарифмом відношення шансів позитивного класу (log-odds) у початкових даних. Таким чином, ансамбль починає роботу не «з нуля», а з апріорної оцінки ймовірності атаки, що дозволяє значно прискорити збіжність на ранніх етапах навчання.

У випадку балансованих даних (приблизно рівна кількість атак і нормальних подій) $F_0 \approx 0$, тобто початкові прогнози розташовані близько до 0.5.

На кожному кроці бустингу будується нове дерево, яке апроксимує помилки попереднього ансамблю. Ці помилки визначаються як антиградієнт функції втрат за поточними передбаченнями, тобто:

$$r_{im} = y_i - \sigma(F_{m-1}(x_i)). \quad (2.17)$$

Тут r_{im} – псевдорезидуал, який показує, наскільки передбачена ймовірність відрізняється від фактичної мітки.

Якщо модель занижує оцінку атаки ($p_i < y_i$), резидуал додатний, і нове дерево має «посилити» прогноз; якщо завищує ($p_i > y_i$), резидуал від'ємний, і дерево має зменшити оцінку.

Цей механізм утворює градієнтну динаміку навчання: кожне дерево спрямоване на зменшення поточного градієнта похибки, поступово наближаючи ансамбль до мінімуму функції втрат.

Після побудови дерева, що групує спостереження за схожістю резидуалів, потрібно визначити оптимальну вагу (зсув) γ_{jm} для кожного листка R_{jm}:

$$\mathrm{R}_{\mathrm{jm}} = \arg\min_{\gamma} \sum_{x_i \in \mathrm{R}_{\mathrm{jm}}} l(y_i, F_{M-1}(x_i)) + \gamma, \tag{2.18}$$

де $l(y,F)$ – функція втрат (у нашому випадку – логістична).

Інтуїтивно це означає: для всіх об'єктів, що потрапили до одного листка дерева, ми шукаємо таку константу γ_{jm}, яка найкраще коригує попередні прогнози.

Вона може бути як додатною (збільшує логіт), так і від'ємною (зменшує логіт) залежно від середнього знаку резидуалів у цьому листку.

Таким чином, побудова кожного дерева включає дві фази:

– створення структури дерева (розбиття за ознаками);

– оптимізацію зсувів γ_{jm} для мінімізації похибки всередині листків.

Аналітичне мінімізування (2.18) для логістичної втрати складне, тому використовують Ньютонівське наближення:

$$\gamma_{\mathrm{jm}} \approx -\frac{\sum_{x_i \in \mathrm{R}_{\mathrm{jm}}} g_i}{\sum_{x_i \in \mathrm{R}_{\mathrm{jm}}} h_i + \lambda}, g_i = p_i^{(m-1)} - y_i, h_i = p_i^{(m-1)}\left(1 - p_i^{(m-1)}\right). \tag{2.19}$$

де:

g_i – перша похідна втрати (градієнт);

h_i – друга похідна (гесій);

λ – малий стабілізаційний коефіцієнт.

Це наближення дає змогу швидко обчислити оптимальні ваги для кожного листка, що значно прискорює процес навчання.

По суті, формула (2.19) визначає, як сильно і в якому напрямку кожен листок коригує прогноз ансамблю, враховуючи як поточну похибку, так і кривину функції втрат.

Щоб побудувати чергове дерево $h_m(x)$ у складі ансамблю, необхідно визначити, за якою ознакою та при якому порозі потрібно розділити навчальні приклади. Цей процес базується на максимізації приросту інформації, тобто мінімізації невизначеності (ентропії) усередині вузлів.

Формально критерій розщеплення визначається як:

$$\Delta H = H(S) - \sum_{v\in\{L,R\}} \frac{|S_v|}{|S|} \cdot H(S_v), \tag{2.20}$$

де:

S – множина прикладів у поточному вузлі;

S_L та S_R – підмножини після розділення за деякою ознакою;

$H(S) - \sum_{c\in\{0,1\}} p_c \ln p_c$ – ентропія вузла.

Інтуїтивно, ΔH вимірює, наскільки чистішими стали підвузли після розщеплення: чим більший приріст інформації, тим корисніше розділення.

У контексті навчання на резидуалах (2.17) критерій може бути узагальнений у вигляді приросту за зменшенням втрати, коли замість ентропії використовується логістична втрата. У реалізації методу швидкого дерева цей критерій розраховується не просто на основі частот, а через агреговані градієнти g_i та гесії h_i, що дозволяє пришвидшити обчислення без втрати точності.

Таким чином, кожне дерево всередині ансамблю будується не випадково, а на основі чітко визначеного математичного принципу – максимізації локального інформаційного виграшу.

Після завершення побудови *M* дерев ансамбль представляє собою суму їхніх внесків на логіт-шкалі. Ця величина визначається як:

$$\hat{F}(x) = \sum_{m=1}^{M} \nu\gamma_{\mathrm{m}} h_m(x) + F_0, \tag{2.21}$$

де:

$h_m(x)$ – прогноз m-го дерева для вхідного вектора x;

γ_m – оптимальна вага дерева;

F_0 – початкове значення ансамблю, визначене на основі;

v – коефіцієнт усадки, що регулює швидкість навчання.

Отже, $\hat{F}(x)$ є сумарним логітом, який інтегрує знання всіх попередніх ітерацій навчання.

Це величина, що містить «колективний досвід» усіх дерев, які послідовно зменшували похибку на основі резидуалів.

На інтуїтивному рівні, якщо певна комбінація ознак часто з'являється в атакувальних сесіях, то послідовність дерев поступово збільшує логіт для таких прикладів, зміщуючи ймовірність до одиниці.

Після обчислення ансамблевого логіту $\hat{F}(x)$ необхідно знову перейти до ймовірнісного подання, щоб інтерпретувати результат з точки зору упевненості системи.

Це здійснюється за допомогою тієї ж сигмоїдної функції:

$$\hat{p}(x) = \sigma\left(\hat{F}(x)\right) = \frac{1}{1+e^{\hat{F}(x)}}. \quad (2.22)$$

Отримане значення $\hat{p}(x)$ належить інтервалу (0,1) і трактується як оцінка ймовірності того, що даний мережевий запис є атакою.

Таким чином, незалежно від складності ансамблю, вихідна інтерпретація залишається ідентичною до логістичної регресії, що дозволяє уніфікувати систему оцінки якості (метрики, пороги, валідацію).

У практичному застосуванні цей підхід має важливу властивість:

навіть якщо ансамбль складається з десятків дерев, його результат – це єдиний скаляр $\hat{p}(x)$, який може бути безпосередньо використаний у правилах безпеки, звітах чи автоматичних тригерах реагування (наприклад, блокування з'єднання при $\hat{p} > 0{,}8$).

2.4 Визначення метрик та критеріїв оцінювання якості роботи моделей

Ефективність побудованих моделей машинного навчання у задачах інформаційної безпеки визначається не лише їхньою здатністю здійснювати

класифікацію мережевих подій, а й точністю, повнотою та стабільністю цих рішень при різних варіаціях даних.

З огляду на критичність пропуску реальної атаки, оцінювання якості моделей повинно базуватись на системі кількісних метрик, що враховують не лише правильність прогнозів, але й співвідношення між виявленими та пропущеними загрозами, а також рівень упевненості системи у власних рішеннях.

У контексті запропонованої математичної моделі вихідними даними для оцінювання є порівняння передбачених класів $\hat{y}_i$ з істинними мітками y_i.

На їхній основі формується матриця невідповідностей, або confusion matrix, що слугує аналітичною основою для розрахунку всіх подальших метрик.

Нехай результати роботи моделі на тестовій множині подано у вигляді матриці:

$$CM = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix}. \quad (2.23)$$

Тут:

TP (*True Positives*) – кількість випадків, коли атаки були правильно ідентифіковані;

FP (*False Positives*) – кількість нормальних з’єднань, помилково віднесених до атак;

FN (*False Negatives*) – кількість атак, які система не розпізнала;

TN (*True Negatives*) – кількість нормальних подій, коректно класифікованих як безпечні.

Матриця невідповідностей дозволяє формалізовано оцінити якість класифікації та забезпечує основу для обчислення точності, повноти, збалансованості та інших похідних метрик.

У сфері інформаційної безпеки особливе значення мають саме елементи FN та FP, оскільки пропущена атака може спричинити суттєві наслідки для системи, тоді як надлишкові хибні тривоги здатні перевантажити інфраструктуру або призвести до ігнорування сигналів захисту.

Загальна ефективність моделі вимірюється показником точності (*Accuracy*):

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN}. \quad (2.24)$$

Показник точності відображає частку коректно класифікованих спостережень серед усіх перевірених. Високе значення точності свідчить про загальну відповідність прогнозів дійсності, проте цей показник є нечутливим до дисбалансу класів.

У випадках, коли кількість нормальних з'єднань у багато разів перевищує кількість атак, навіть тривіальний класифікатор, який завжди прогнозує «норму», може демонструвати високу точність. Тому для адекватної оцінки продуктивності у задачах кіберзахисту необхідно застосовувати більш інформативні метрики – *Precision* і *Recall*.

Метрики точності передбачення (*Precision*) і повноти виявлення (*Recall*) деталізують різні аспекти роботи класифікатора:

$$Precision = \frac{TP}{TP+FP},\ Recall = \frac{TP}{TP+FN}. \quad (2.25)$$

Precision показує, яку частку серед усіх випадків, визначених системою як атаки, становлять справжні загрози. Це – показник надійності попереджень: високий рівень *Precision* означає, що система не генерує зайвих хибних тривог і довіряти її сигналам можна з високою ймовірністю.

Recall, натомість, характеризує чутливість або здатність моделі виявляти реальні атаки. Високе значення *Recall* свідчить про мінімізацію кількості пропущених загроз.

Однак надмірне збільшення чутливості може знизити точність, тому між цими показниками завжди існує компроміс.

Для одночасного врахування точності та повноти використовується *F1*-міра, що визначається як гармонійне середнє між ними:

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision+Recall}. \quad (2.26)$$

Ця метрика дозволяє збалансовано оцінити ефективність моделі, надаючи однакову вагу як здатності виявляти атаки, так і точності їх класифікації.

Завдяки своїй стійкості до дисбалансу класів *F1*-міра є найбільш інформативною у ситуаціях, коли кількість позитивних випадків (атак) є значно меншою за кількість негативних (нормального трафіку).

F1 вказує на оптимальний баланс між мінімізацією пропущених загроз і скороченням хибних спрацювань, що особливо важливо для автоматизованих систем моніторингу.

Для поглибленого аналізу якості класифікації використовується логарифмічна втрата (*LogLoss*), або перехресна ентропія, яка оцінює відповідність передбачених імовірностей реальним міткам:

$$LogLoss = -\frac{1}{n_{test}}\sum_{i=1}^{n_{test}}[y_i \ln \hat{p}(x_i) + (1 - y_i)\ln(1 - \hat{p}(x_i))]. \quad (2.27)$$

Ця функція втрат надає змогу оцінити каліброваність моделі, тобто наскільки впевненість прогнозів узгоджується з реальністю.

Якщо система помиляється з високим рівнем упевненості (наприклад, передбачає для реальної атаки $\hat{p} = 0{,}05$), штраф у *LogLoss* буде суттєвим.

Натомість помилки з низькою впевненістю (коли прогноз був невизначеним, наприклад 0.55) караються значно менше.

Таким чином, *LogLoss* забезпечує імовірнісну оцінку якості, яка є важливою у випадках, коли рішення приймаються автоматично – наприклад, при динамічному блокуванні підозрілих з'єднань чи адаптивному регулюванні порогів спрацювання.

Отже, система оцінювання, що базується на метриках (2.23 – 2.27), забезпечує науково обґрунтований підхід до аналізу ефективності моделей машинного навчання у сфері інформаційної безпеки. Її застосування дозволяє не лише вимірювати якість класифікації, а й здійснювати інтерпретацію результатів з точки зору ризику, надійності та стійкості системи до помилок. Завдяки цьому формалізованому апарату стає можливим об'єктивне порівняння різних алгоритмів навчання, визначення їх оптимальних параметрів і побудова комплексних систем захисту, де рішення приймаються на основі узгоджених, математично вивірених критеріїв якості.

Висновки до другого розділу

У межах цього розділу сформовано методичні основи застосування технологій штучного інтелекту в інформаційній безпеці, спрямовані на побудову ефективних моделей виявлення кібератак у мережевому трафіку. Проведено оцінку та порівняння провідних методів машинного навчання – градієнтного бустингу, швидкого дерева та логістичної регресії – які є базовими інструментами сучасних інтелектуальних систем безпеки. Для кожного з них розглянуто принцип функціонування, проаналізовано структуру роботи та наведено ключові переваги й недоліки, що дозволило визначити особливості їх застосування в задачах класифікації аномального трафіку й сформувати основу для побудови математичної моделі системи виявлення кібератак.

Для навчання та тестування моделей обґрунтовано вибір набору даних CyberFedDefender, що містить реалістичні ознаки мережевих потоків. Попередній аналіз цього датасету охоплював кількісні та візуальні методи оцінки структури даних: побудовано теплову карту кореляцій між ознаками, досліджено динаміку трафіку за протоколами, виконано просторову візуалізацію у 3D- та 2D-форматах для виявлення кластеризації спостережень, а також проаналізовано джерела трафіку та розподіл даних за мітками атак..

У результаті проведеної роботи розроблено узагальнену математичну модель класифікації мережевих загроз, яка описує повний цикл функціонування інтелектуальної системи – від підготовки та нормалізації даних до побудови прогнозу й обчислення показників якості. Модель передбачає варіативність алгоритмів навчання, що дозволяє здійснити об’єктивне порівняння їхньої ефективності. Для забезпечення достовірності оцінювання було визначено систему метрик які комплексно відображають як точність класифікації, так і здатність моделі мінімізувати хибні спрацьовування.

Отримані результати створюють підґрунтя для наступного розділу, у якому буде представлено програмну реалізацію побудованої системи, проведено навчання моделей, експериментальні тести на реальних даних та оцінено ефективність застосованих алгоритмів у задачах виявлення кібератак.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

3.1 Вибір мови програмування та бібліотек для реалізації системи

Розроблення інтелектуальної системи виявлення кібератак потребує ретельного вибору мови програмування та відповідного набору бібліотек, здатних забезпечити стабільну інтеграцію методів машинного навчання, ефективну роботу з даними та високу продуктивність у реальному часі. На етапі технічного обґрунтування доцільно розглянути найпоширеніші мови, які активно застосовуються у сфері аналітики безпеки – Python, Java та C#, – кожна з яких має власні переваги, екосистему бібліотек і рівень сумісності з сучасними AI-інструментами. Їх порівняльна характеристика дозволяє визначити оптимальний технологічний стек для реалізації системи моніторингу та виявлення загроз.

Python є найпоширенішою мовою у сфері штучного інтелекту та оброблення даних завдяки своїй гнучкості, простому синтаксису та потужній екосистемі бібліотек – таких як TensorFlow, PyTorch, Scikit-learn і Pandas [45]. Вона забезпечує швидке прототипування моделей машинного навчання й інтеграцію з аналітичними інструментами. Python широко використовується для створення систем виявлення аномалій, прогнозування кібератак і побудови алгоритмів класифікації трафіку.

Java характеризується високою продуктивністю, кросплатформеністю та стабільністю, що робить її придатною для розробки масштабованих корпоративних систем безпеки [46]. Завдяки бібліотекам Weka, Deeplearning4j і MOA Java дозволяє реалізовувати машинне навчання у розподілених середовищах. Її сувора типізація й надійна система керування пам’яттю забезпечують безпечність виконання програм у критично важливих додатках.

C# вирізняється високою сумісністю з технологіями Microsoft і платформою .NET, що сприяє розробленню інтерактивних додатків із графічним інтерфейсом користувача [47]. Мова підтримує бібліотеку ML.NET, яка дозволяє створювати, навчати й тестувати моделі машинного навчання безпосередньо в середовищі

Visual Studio. C# поєднує об’єктно-орієнтований підхід із можливостями швидкої інтеграції аналітичних модулів, що робить її ефективним вибором для реалізації систем інформаційної безпеки на базі AI-технологій.

Для забезпечення обґрунтованого вибору серед доступних інструментів доцільно здійснити порівняльний аналіз розглянутих мов програмування за основними характеристиками, які впливають на ефективність створення систем виявлення кібератак. Узагальнені результати наведено в табл. 3.1, що дає змогу оцінити потенціал кожної мови в контексті розроблення інтелектуальних систем інформаційної безпеки.

Таблиця 3.1

Порівняльна характеристика мов програмування

№	Характеристика	Python	Java	C#
1	Простота синтаксису та швидкість розробки	Висока	Середня	Висока
2	Продуктивність виконання (кількість операцій за секунду, відн. од.)	0.8	1.0	1.1
3	Середній час навчання моделі на вибірці (сек.)	120	95	90
4	Використання оперативної пам’яті (МБ)	450	380	310
5	Можливості побудови GUI	Обмежені	Середні	Розширені
6	Підтримка бібліотек машинного навчання	Дуже широка	Обмежена	Оптимізована (ML.NET)
7	Інтеграція з корпоративними системами	Середня	Висока	Висока
9	Масштабованість у великих системах	Середня	Висока	Висока
10	Стійкість до помилок виконання та типізація	Динамічна, середня	Статична, висока	Статична, висока
11	Можливість кросплатформенного розгортання	Висока	Висока	Висока (.NET Core)
12	Графічна інтеграція аналітичних модулів	Обмежена	Середня	Розширена

Виходячи з проведеного аналізу, мову програмування C# обрано як основний інструмент для реалізації системи виявлення кібератак, оскільки вона поєднує

високу продуктивність, надійну типізацію та широкі можливості інтеграції з компонентами інформаційної інфраструктури підприємства. Платформа .NET забезпечує стабільне середовище для побудови як аналітичних модулів, так і зручного графічного інтерфейсу користувача, що є важливою складовою при розробленні інтерактивних систем моніторингу безпеки. Завдяки бібліотеці ML.NET C# дозволяє реалізовувати повний цикл машинного навчання – від підготовки даних до побудови та оцінювання моделей – без потреби у сторонніх середовищах. Крім того, мова відзначається оптимальним балансом між швидкодією, ефективним використанням пам’яті та зручністю відлагодження коду. Сукупність цих характеристик робить C# технологічно доцільним вибором для створення надійної, масштабованої та функціонально насиченої системи інформаційної безпеки.

Оскільки для розроблення системи виявлення кібератак обрано мову програмування C#, доцільним є аналіз бібліотек машинного навчання, сумісних із платформою .NET, які забезпечують ефективну роботу з даними, навчання моделей і їх подальшу інтеграцію в програмне середовище. Найбільш відомими та функціонально придатними для реалізації даного проєкту є бібліотеки ML.NET, Accord.NET та Encog, кожна з яких має власну спеціалізацію й набір переваг для побудови інтелектуальних систем безпеки.

ML.NET – це офіційна бібліотека від Microsoft, створена для розроблення моделей машинного навчання безпосередньо в середовищі .NET [49]. Вона підтримує основні типи задач – класифікацію, регресію, кластеризацію, прогнозування та детекцію аномалій – і дозволяє інтегрувати моделі безпосередньо у WinForms або веб-додатки. ML.NET забезпечує високу швидкодію, просту побудову конвеєрів оброблення даних і сумісність із моделями, експортованими у формат ONNX, що робить її оптимальним вибором для промислових застосувань.

Accord.NET – це розширена бібліотека з відкритим кодом, орієнтована на наукові обчислення, оброблення сигналів, зображень та реалізацію класичних алгоритмів машинного навчання [49]. Вона містить широкий набір інструментів

для статистичного аналізу, роботи з часовими рядами та класифікації, включно з методами SVM, KNN, дерева рішень і нейронними мережами.

Encog – це бібліотека для побудови та навчання нейронних мереж, оптимізована для роботи в середовищі .NET та Java [50, 51]. Вона підтримує різноманітні архітектури, зокрема багатошарові персептрони, рекурентні та радіальні мережі, що дозволяє реалізовувати глибше моделювання складних залежностей у даних.

Для формування обґрунтованого вибору розглянемо ML.NET, Accord.NET та Encog за основними критеріями продуктивності, гнучкості, простоти використання та підтримки сучасних алгоритмів штучного інтелекту. Узагальнені результати порівняння подано в табл. 3.2.

Таблиця 3.2

Порівняльна характеристика бібліотек машинного навчання для C#

№	Характеристика	ML.NET	Accord.NET	Encog
1	Підтримка типових задач (класифікація, регресія, кластеризація)	Повна	Повна	Часткова
2	Інтеграція з середовищем .NET та Visual Studio	Висока (офіційна підтримка Microsoft)	Середня	Висока
3	Швидкодія при роботі з великими наборами даних	Висока	Середня	Висока
4	Простота побудови конвеєрів навчання	Висока	Середня	Низька
5	Підтримка нейронних мереж	Обмежена (через ONNX Runtime)	Середня	Розширена
6	Можливість експорту/імпорту моделей	Так (ONNX, ZIP)	Частково	Так
7	Наявність засобів попередньої обробки даних	Так (DataView API, трансформери)	Так	Частково
8	Доступність документації та прикладів	Дуже висока	Висока	Середня
9	Розширення функціоналу за рахунок зовнішніх модулів	Підтримується (TensorFlow, ONNX)	Частково	Обмежено
10	Зручність інтеграції з GUI-додатками	Висока	Середня	Середня

Виходячи з проведеного порівняльного аналізу, для реалізації системи виявлення кібератак обрано бібліотеку ML.NET, яка найповніше відповідає вимогам до продуктивності, гнучкості та інтеграційних можливостей у середовищі .NET. Її архітектура дозволяє створювати повноцінні конвеєри машинного навчання – від завантаження та підготовки даних до навчання, тестування й збереження моделі. Високий рівень сумісності з Visual Studio, зручний API та підтримка формату ONNX забезпечують можливість подальшого розширення системи й використання попередньо навчених моделей. Окрім того, ML.NET відзначається стабільністю, офіційною підтримкою Microsoft і добре документованими прикладами, що суттєво спрощує розроблення промислових рішень у сфері інформаційної безпеки.

3.2 Реалізація застосунку для навчання та оцінювання моделей

Реалізація застосунку для навчання та оцінювання моделей у середовищі C# з використанням бібліотеки ML.NET передбачає побудову структур даних, що забезпечують коректне завантаження, підготовку та оброблення вхідної інформації з набору даних CyberFedDefender. На цьому етапі формуються класи, які описують структуру записів мережевого трафіку, зокрема значення ознак, необхідних для побудови моделі класифікації. Одним із ключових елементів архітектури є клас NetworkRecord, який визначає схему даних для подальшого навчання та тестування моделей виявлення кібератак. Його використання дає змогу ML.NET автоматично зв’язати стовпці CSV-файлу з відповідними властивостями об’єкта.

На рис. 3.1 наведено фрагмент коду класу NetworkRecord, що описує набір характеристик мережевого трафіку. Клас NetworkRecord слугує основою для побудови вхідних об’єктів, які передаються до навчального конвеєра ML.NET. Кожен атрибут позначено анотацією [LoadColumn], яка вказує номер колонки у вхідному CSV-файлі, з якого здійснюється імпорт даних. До складу класу входять як числові параметри (наприклад, Packet_Length, Bytes_Sent, Flow_Bytes_per_s), так і категоріальні (зокрема, Protocol, Flags, Attack_Type). Така структура дозволяє моделі обробляти як кількісні, так і якісні ознаки мережевої активності.

```
public class NetworkRecord {
    0 references
    [LoadColumn(0)] public string Timestamp { get; set; }
    0 references
    [LoadColumn(1)] public string Source_IP { get; set; }
    0 references
    [LoadColumn(2)] public string Destination_IP { get; set; }
    6 references
    [LoadColumn(3)] public string Protocol { get; set; }
    8 references
    [LoadColumn(4)] public float Packet_Length { get; set; }
    8 references
    [LoadColumn(5)] public float Duration { get; set; }
    8 references
    [LoadColumn(6)] public float Source_Port { get; set; }
    8 references
    [LoadColumn(7)] public float Destination_Port { get; set;
    8 references
    [LoadColumn(8)] public float Bytes_Sent { get; set; }
    8 references
    [LoadColumn(9)] public float Bytes_Received { get; set; }
    6 references
    [LoadColumn(10)] public string Flags { get; set; }
    8 references
    [LoadColumn(11)] public float Flow_Packets_per_s { get; se
    8 references
    [LoadColumn(12)] public float Flow_Bytes_per_s { get; set; }
    8 references
    [LoadColumn(13)] public float Avg_Packet_Size { get; set; }
    8 references
    [LoadColumn(14)] public float Total_Fwd_Packets { get; set; }
    8 references
    [LoadColumn(15)] public float Total_Bwd_Packets { get; set; }
    8 references
    [LoadColumn(16)] public float Fwd_Header_Length { get; set; }
    8 references
    [LoadColumn(17)] public float Bwd_Header_Length { get; set; }
    8 references
    [LoadColumn(18)] public float Sub_Flow_Fwd_Bytes { get; set; }
    8 references
    [LoadColumn(19)] public float Sub_Flow_Bwd_Bytes { get; set; }
    8 references
    [LoadColumn(20)] public float Inbound { get; set; }
    4 references
    [LoadColumn(21)] public string Attack_Type { get; set; }
    5 references
    [LoadColumn(22)] public float Label { get; set; }
}
```

Рис. 3.1 Код класу вхідних даних NetworkRecord

Окрім основних характеристик, клас містить поля, що описують напрямок трафіку (Inbound), статистику потоків (Total_Fwd_Packets, Total_Bwd_Packets, Fwd_Header_Length, Bwd_Header_Length) та узагальнену мітку (Label), яка визначає тип запису – нормальний трафік або атаку. Завдяки чіткому розмежуванню вхідних і цільових змінних цей клас забезпечує правильну побудову навчального конвеєра, у якому передбачено етапи нормалізації, перетворення текстових атрибутів у числові та подальшої класифікації.

Клас ThreatPrediction використовується для представлення результатів роботи навченої моделі після виконання прогнозування (рис. 3.2). Його структура визначає формат вихідних даних, що повертаються під час класифікації мережевого трафіку, зокрема – чи є конкретний запис потенційною кіберзагрозою.

```
public class ThreatPrediction {
    // Результат: true = загроза
    3 references
    [ColumnName("PredictedLabel")] public bool PredictedLabel { get; set; }
    3 references
    public float Probability { get; set; }  // Ймовірність прогнозу (0-1)
    3 references
    public float Score { get; set; }        // Вага/рейтинг прогнозу
}
```

Рис. 3.2 Код класу «ThreatPrediction»

Поле PredictedLabel містить логічне значення, де true відповідає виявленню аномальної активності або атаки, а false – нормальному трафіку. Атрибут [ColumnName("PredictedLabel")] встановлює зв’язок із вихідною колонкою, яку формує ML.NET під час прогнозу. Додаткові властивості Probability та Score

забезпечують кількісну оцінку достовірності результату: перша відображає ймовірність віднесення об’єкта до класу атаки в межах від 0 до 1, а друга – числовий показник інтенсивності або впевненості моделі у прийнятому рішенні. Завдяки цьому класу результати прогнозу можна не лише інтерпретувати бінарно, а й аналізувати рівень загрози, що підвищує аналітичну цінність системи для фахівців з інформаційної безпеки.

Наведений фрагмент коду на рис. 3.3 реалізує обробник події натискання кнопки OpenBtn, яка відповідає за відкриття файлу з локальної системи користувача. Після активації кнопки створюється діалогове вікно типу OpenFileDialog, що дозволяє обрати вхідний файл для подальшої обробки в системі.

```
private void OpenBtn_Click(object sender, EventArgs e) {
  using (OpenFileDialog openDlg = new OpenFileDialog()) {
    openDlg.Filter = "CSV-файли (*.csv)|*.csv|Усі файли (*.*)|*.*";
    openDlg.FilterIndex = 1;
    openDlg.RestoreDirectory = true;

    if (openDlg.ShowDialog() != DialogResult.OK) return;

    _Path = openDlg.FileName;
    FileNameTBox.Text = _Path;
```

Рис. 3.3 Відкриття файлу з локальної системи користувача

За допомогою властивості Filter користувачу пропонується відфільтрований вибір лише файлів із розширенням .csv, що є стандартним форматом для наборів даних, а також надається можливість переглянути всі типи файлів у разі потреби. Встановлення параметра RestoreDirectory = true гарантує повернення до початкової директорії після закриття діалогу, забезпечуючи зручність повторного вибору. Якщо користувач підтверджує вибір, шлях до вибраного файлу зберігається у змінній _Path, а його ім’я відображається у текстовому полі FileNameTBox, що забезпечує візуальне підтвердження завантаження даних у графічному інтерфейсі застосунку.

На рис. 3.4 наведено фрагмент коду у якому реалізовано початковий етап підготовки даних для навчання моделі машинного навчання в середовищі ML.NET. Спочатку створюється об’єкт MLContext, який є базовим компонентом усіх

ML.NET-проєктів і відповідає за керування життєвим циклом моделей, журналювання процесів та відтворюваність результатів завдяки встановленому параметру seed: 123. Далі за допомогою методу LoadFromTextFile відбувається завантаження даних із CSV-файлу, шлях до якого задається змінною _Path, у strongly-typed об’єкт IDataView, що використовує клас NetworkRecord як опис структури даних. Це дозволяє системі автоматично інтерпретувати стовпці CSV-файлу як властивості класу з відповідними типами.

```
// 1) MLContext
mlContext = new MLContext(seed: 123);
// 2) Завантаження CSV у strongly-typed IDataView
dataView = mlContext.Data.LoadFromTextFile<NetworkRecord>(
                path: _Path, separatorChar: ',', hasHeader: true);
// 3) Коротка статистика по мітці (0/1 як float у CSV)
var labelCounts = mlContext.Data.CreateEnumerable<NetworkRecord>(dataView, false)
                                .GroupBy(r => r.Label)
                                .Select(g => new { Label = g.Key, Cnt = g.Count() })
                                .OrderBy(x => x.Label)
                                .ToList();
RaportTBox.AppendText("Кількість взірців за міткою (float 0/1 у CSV):\r\n");
foreach (var lc in labelCounts)
  RaportTBox.AppendText($"Label={lc.Label}: {lc.Cnt}\r\n");
```

Рис. 3.4 Етап підготовки даних для навчання моделі машинного навчання

Після завантаження даних виконується підрахунок кількості записів за мітками класифікації, де значення Label рівне 0 відповідає нормальному трафіку, а Label рівне 1 – виявленим кібератакам. Для цього дані конвертуються у колекцію об’єктів NetworkRecord, групуються за ознакою Label та підраховується кількість елементів у кожній групі. Отримані результати виводяться у текстове поле RaportTBox, що слугує елементом інтерфейсу для відображення поточної статистики. Таким чином, цей код забезпечує початкову валідацію даних і дозволяє швидко перевірити збалансованість класів перед подальшим навчанням моделей.

У фрагменті коду на рис. 3.5 визначається набір ознак, які використовуються для побудови моделі машинного навчання в рамках системи виявлення кібератак. Всі вхідні дані поділено на дві групи – числові (numeric) та категоріальні (categorical) – залежно від типу інформації, яку вони містять. До числових ознак віднесено параметри, що описують кількісні характеристики мережевого трафіку, зокрема довжину пакета, обсяг переданих байтів, швидкість потоку, кількість

пакетів у прямому та зворотному напрямках, а також технічні показники з'єднання. Ці параметри безпосередньо впливають на поведінкову модель потоку й використовуються для визначення аномалій.

```
string[] numeric = {
    nameof(NetworkRecord.Packet_Length),
    nameof(NetworkRecord.Duration),
    nameof(NetworkRecord.Source_Port),
    nameof(NetworkRecord.Destination_Port),
    nameof(NetworkRecord.Bytes_Sent),
    nameof(NetworkRecord.Bytes_Received),
    nameof(NetworkRecord.Flow_Packets_per_s),
    nameof(NetworkRecord.Flow_Bytes_per_s),
    nameof(NetworkRecord.Avg_Packet_Size),
    nameof(NetworkRecord.Total_Fwd_Packets),
    nameof(NetworkRecord.Total_Bwd_Packets),
    nameof(NetworkRecord.Fwd_Header_Length),
    nameof(NetworkRecord.Bwd_Header_Length),
    nameof(NetworkRecord.Sub_Flow_Fwd_Bytes),
    nameof(NetworkRecord.Sub_Flow_Bwd_Bytes),
    nameof(NetworkRecord.Inbound)
};
string[] categorical = { nameof(NetworkRecord.Protocol), nameof(NetworkRecord.Flags) };
```

Рис. 3.5 Визначення набору ознак для побудови моделі

Категоріальні ознаки – Protocol і Flags – описують якісні властивості мережевих з'єднань, такі як тип протоколу (TCP, UDP, ICMP) та стан передачі даних на основі TCP-прапорів (SYN, ACK, FIN тощо). Включення таких змінних дозволяє моделі не лише працювати з кількісними значеннями, а й враховувати контекст комунікацій між вузлами мережі. Така класифікація ознак є підготовчим етапом перед побудовою конвеєра оброблення даних, у якому числові параметри нормалізуються, а категоріальні – перетворюються у числові вектори для подальшого навчання моделі.

У наведеному на рис. 3.6 фрагменті коду реалізовано етап попередньої обробки даних (preprocessing), який є необхідним для підготовки вхідного набору до навчання моделі машинного навчання в середовищі ML.NET. На початку здійснюється перетворення стовпця Label, який у вихідному датасеті має тип float, у логічний формат bool. Це потрібно для того, щоб алгоритм класифікації міг працювати з бінарною міткою, де true відповідає наявності кібератаки, а false – нормальному трафіку.

```
var preprocessing =
    mlContext.Transforms.Conversion.ConvertType(
        outputKind: DataKind.Boolean,
        inputColumnName: "Label",
        outputColumnName: "Label")
    .Append(mlContext.Transforms.Categorical.OneHotEncoding("Protocol_OHE", "Protocol"))
    .Append(mlContext.Transforms.Categorical.OneHotEncoding("Flags_OHE", "Flags"))
    .Append(mlContext.Transforms.Concatenate(
        "Features",
        MLHelper.Combine(numeric, new[] { "Protocol_OHE", "Flags_OHE" })))
    .Append(mlContext.Transforms.NormalizeMinMax("Features"));
```

Рис. 3.6 Попередня обробка даних

Далі виконується кодування категоріальних ознак за допомогою методу OneHotEncoding, який перетворює текстові значення стовпців Protocol та Flags у числові вектори. Таке представлення дозволяє моделі сприймати різні протоколи зв’язку або стани з’єднання як окремі незалежні параметри. Після цього всі числові та закодовані ознаки об’єднуються у єдиний вектор під назвою Features, який використовується як вхідний простір для навчання класифікатора. Завершальним етапом є нормалізація значень методом MinMax, що масштабує всі показники в межах від 0 до 1. Це гарантує рівнозначний вплив кожної ознаки на процес навчання, усуває дисбаланс між параметрами з різними одиницями вимірювання та підвищує стабільність роботи моделі.

Фрагмент коду на рис. 3.7 створює та налаштовує метод швидкого дерева, який використовується для виконання задачі бінарної класифікації у системі виявлення кібератак. Алгоритм належить до класу градієнтних дерев рішень і є одним із найефективніших методів, реалізованих у бібліотеці ML.NET, завдяки здатності виявляти складні нелінійні залежності між вхідними ознаками. Під час ініціалізації вказується назва стовпця із цільовими мітками (Label) та набір ознак (Features), які використовуються для навчання моделі.

```
var fastTree = mlContext.BinaryClassification.Trainers.FastTree(
    labelColumnName: "Label",
    featureColumnName: "Features",
    numberOfLeaves: 20,
    numberOfTrees: 100,
    minimumExampleCountPerLeaf: 10);
```

Рис. 3.7 Створення та налаштування методу швидкого дерева

Основні параметри алгоритму визначають глибину і структуру ансамблю дерев. Значення numberOfLeaves рівне 20 обмежує максимальну кількість листків у кожному дереві, що дозволяє уникнути перенавчання. Параметр numberOfTrees рівний 100 задає кількість дерев у моделі, забезпечуючи баланс між точністю класифікації та часом навчання. Останній параметр minimumExampleCountPerLeaf рівний 10 встановлює мінімальну кількість прикладів у кожному листку дерева, що сприяє підвищенню узагальнювальної здатності моделі. У сукупності ці налаштування дають змогу отримати стійку модель, здатну ефективно розпізнавати шкідливу активність у мережевому трафіку навіть у складних і різнорідних умовах.

Наведений фрагмент коду на рис. 3.8 реалізує етап розподілу набору даних на тренувальну та тестову вибірки, що є обов'язковим кроком під час побудови моделей машинного навчання. За допомогою методу TrainTestSplit дані поділяються у співвідношенні 80/20, де 80% використовується для навчання моделі, а 20% – для перевірки її якості на невідомих даних. Встановлення фіксованого параметра seed: 123 гарантує відтворюваність результатів при повторному виконанні експерименту. Отримані підмножини даних зберігаються у змінних trainData та testData, які надалі використовуються відповідно для навчання моделі та її тестування.

```
var split = mlContext.Data.TrainTestSplit(dataView, testFraction: 0.2, seed: 123);
var trainData = split.TrainSet;
var testData = split.TestSet;

RaportTBox.AppendText("\r\n[i] Навчання FastTree…\r\n");
Application.DoEvents();
```

Рис. 3.8 Розподіл набору даних на тренувальну та тестову вибірки

Після поділу даних у текстове поле RaportTBox виводиться повідомлення про початок процесу тренування моделі методом швидкого дерева, що виконує класифікацію мережевого трафіку. Виклик методу «DoEvents» дозволяє оновити інтерфейс користувача, забезпечуючи його чутливість під час виконання обчислень. Таким чином, цей код не лише готує дані до навчання, а й створює зручний механізм для інформування користувача про хід виконання обчислювальних процесів у реальному часі.

Код на рис. 3.9 реалізовує відображення основних метрик оцінювання якості моделі, навченої за допомогою методу швидкого дерева. Після завершення етапу тестування результати роботи класифікатора виводяться у текстове поле RaportTBox, що виконує роль інформаційного журналу у графічному інтерфейсі системи.

```
RaportTBox.AppendText("\r\n=== Загальні метрики (FastTree) ===\r\n");
RaportTBox.AppendText($"Accuracy  : {metrics.Accuracy:P2}\r\n");
RaportTBox.AppendText($"AUC (ROC) : {metrics.AreaUnderRocCurve:P2}\r\n");
RaportTBox.AppendText($"F1-Score  : {metrics.F1Score:P2}\r\n");
RaportTBox.AppendText($"Precision : {metrics.PositivePrecision:P2}\r\n");
RaportTBox.AppendText($"Recall    : {metrics.PositiveRecall:P2}\r\n");
```

Рис. 3.9 Відображення основних метрик оцінювання якості моделі

Зокрема, показник Accuracy відображає загальну частку правильно класифікованих прикладів, тоді як AUC (Area Under ROC Curve) характеризує здатність моделі розрізняти класи нормального та шкідливого трафіку. Метрика F1-Score є гармонійним середнім між точністю (Precision) і повнотою (Recall), що дозволяє комплексно оцінити якість класифікації у випадках із нерівномірним розподілом класів. Вивід кожного показника у відсотковому форматі (P2) забезпечує наочність і спрощує порівняння результатів між різними експериментами або моделями.

Наведений фрагмент коду на рис. 3.10 реалізує обчислення та відображення матриці помилок (confusion matrix) для оцінювання якості класифікаційної моделі після її тестування. Після застосування навченої моделі trainedModel до тестових даних формується набір прогнозів за допомогою методу Transform, а потім ці результати перетворюються у колекцію об’єктів ScoredWithLabel, яка містить як передбачені, так і фактичні мітки класів.

```
var predictions = trainedModel.Transform(testData);
var scored = mlContext.Data.CreateEnumerable<ScoredWithLabel>(predictions, false).ToList();
int tp = scored.Count(p => p.PredictedLabel && p.Label);
int tn = scored.Count(p => !p.PredictedLabel && !p.Label);
int fp = scored.Count(p => p.PredictedLabel && !p.Label);
int fn = scored.Count(p => !p.PredictedLabel && p.Label);
RaportTBox.AppendText("\r\n=== Матриця помилок ===\r\n");
RaportTBox.AppendText($"TP: {tp}  |  TN: {tn}\r\nFP: {fp}  |  FN: {fn}\r\n");
```

Рис. 3.10 Обчислення та відображення матриці помилок

Також здійснюється підрахунок основних складових матриці помилок: TP (True Positive) – кількість випадків, коли атака була правильно виявлена; TN (True Negative) – кількість коректно визначених нормальних з’єднань; FP (False Positive) – кількість хибних спрацьовувань, коли звичайний трафік помилково класифіковано як загрозу; FN (False Negative) – випадки, коли модель не виявила наявну атаку. Отримані значення виводяться у текстове поле RaportTBox, що дозволяє користувачу побачити деталізовану структуру помилок класифікації.

На наступному етапі було реалізовано програмну підтримку для інших методів машинного навчання, зокрема градієнтного бустингу та логістичної регресії, які використовуються для порівняння результатів класифікації з алгоритмом швидкого дерева. Аналогічно до попередньої реалізації, побудова конвеєра виконувалася на основі єдиної структури оброблення даних, однак з адаптацією параметрів під особливості кожного методу. У випадку градієнтного бустингу ключовим елементом стала інтеграція алгоритму LightGBM, що забезпечує високу швидкодію та точність за рахунок ітераційного уточнення похибки попередніх моделей.

У наведеному на рис. 3.11 фрагменті коду створюється екземпляр тренера градієнтного бустингу, який ініціалізує процес навчання моделі бінарної класифікації на основі узагальненого ансамблю дерев рішень.

```
var lightGbm = mlContext.BinaryClassification.Trainers.LightGbm(
    labelColumnName: "Label",
    featureColumnName: "Features"
);
```

Рис. 3.11 Створення екземпляру тренера градієнтного бустингу

Задано дві основні колонки – Label, що містить мітки класів (0 – нормальний трафік, 1 – кібератака), та Features, яка об’єднує усі числові та категоріальні ознаки, підготовлені на етапі препроцесингу. Метод градієнтного бустингу використовує механізм ітеративного оновлення моделі, поступово зменшуючи помилки класифікації, що дозволяє ефективно працювати з великими й нерівномірно збалансованими наборами даних. Завдяки внутрішній оптимізації, метод

забезпечує високу швидкість навчання без суттєвої втрати точності, що робить його придатним для побудови систем виявлення кібератак у реальному часі.

Подібно до реалізації конвеєра для градієнтного бустингу, було створено конвеєр для навчання моделі на основі логістичної регресії, який використовує алгоритм SDCA. Цей метод реалізований у бібліотеці ML.NET як ефективний оптимізаційний підхід для задач бінарної класифікації. Його головна перевага полягає у швидкому збіжному навчанні навіть на великих обсягах даних, що містять велику кількість числових і категоріальних ознак.

У наведеному на рис. 3.12 фрагменті коду ініціалізується тренер логістичної регресії, який працює з колонками Label та Features, визначеними раніше в етапі препроцесингу.

```
var sdca = mlContext.BinaryClassification.Trainers.SdcaLogisticRegression(
                labelColumnName: "Label",
                featureColumnName: "Features");
```

Рис. 3.12 Створення екземпляру тренера логістичної регресії

Цей метод навчає модель шляхом мінімізації логістичної втрати, що дозволяє знаходити оптимальні вагові коефіцієнти для кожної ознаки, зберігаючи при цьому інтерпретованість результатів. На відміну від деревних методів, логістична регресія формує лінійну гіперплощину розділення між класами, що робить її зручною для початкового аналізу впливу атрибутів трафіку. Використання SDCA у поєднанні з нормалізованими ознаками забезпечує високу стабільність навчання, низьку варіативність результатів і чітке розуміння ролі кожної змінної у процесі класифікації мережевого трафіку.

Наведений фрагмент коду на рис. 3.13 відповідає за завершальний етап роботи з навченою моделлю – її збереження, реєстрацію та документування у базі даних. Метод SaveBtn_Click викликається при натисканні відповідної кнопки збереження у графічному інтерфейсі застосунку. На початку здійснюється перевірка коректності введених даних за допомогою методу IsDataEnteringCorrect, який гарантує, що усі необхідні поля форми заповнені. Далі формується унікальне ім’я файлу моделі з використанням методу GenerateFileName, після чого

визначається локальний шлях до каталогу проєкту, у якому буде розміщено збережений файл.

```
private void SaveBtn_Click(object sender, EventArgs e) {
  if (!IsDataEnteringCorrect()) return;     // ваша перевірка
  string pathName = @"\\teach\\" + GenerateFileName() + ".zip";
  string localProj = Path.GetDirectoryName(
                     System.Reflection.Assembly.GetExecutingAssembly().Location);
  // зберегти в БД назву й шлях моделі
  _ModelsProvider.InsertModels(ModelsNamesTBox.Text, "Логістична регресія", pathName);
  // фізичне збереження .zip
  mlContext.Model.Save(trainedModel, dataView.Schema, localProj + pathName);
  // логування, очищення форми
  _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
      "Було навчено модель " + ModelsNamesTBox.Text, DateTime.Now);
  ClearAllData();
  MessageBox.Show("Дані успішно збережено!");
}
```

Рис. 3.13 Метод для зберігання моделі

Збереження моделі здійснюється двома способами: логічним і фізичним. На логічному рівні інформація про модель – її назва, тип та шлях до файлу – додається до бази даних через метод InsertModels, що забезпечує можливість подальшого відстеження створених моделей у системі. Фізичне збереження виконується за допомогою методу Save, який експортує навчений об’єкт trainedModel разом із його схемою у ZIP-файл. Після цього створюється запис у журналі подій через InsertLogs, що фіксує факт навчання та збереження моделі користувачем. Завершальним етапом є очищення форми методом ClearAllData та виведення повідомлення Show, яке інформує користувача про успішне завершення операції.

У фрагменті на рис. 3.14 коду реалізовано механізм завантаження раніше навченої моделі машинного навчання для її подальшого використання у прогнозуванні.

```
private void LoadData(string filePathRelative) {
  // Якщо у БД збережено відносний шлях (як у вас) – будуймо повний:
  string localPath = Application.StartupPath + filePathRelative;
  // 1) Завантаження моделі
  DataViewSchema modelSchema;
  ITransformer model = mlContext.Model.Load(localPath, out modelSchema);
  // 2) PredictionEngine для наших типів
  predictionEngine =
    mlContext.Model.CreatePredictionEngine<NetworkRecord, ThreatPrediction>(model);
}
```

Рис. 3.14 Завантаження моделі

Метод LoadData приймає відносний шлях до файлу моделі, який зберігається у базі даних після етапу тренування. Оскільки у базі міститься лише частина шляху, система динамічно формує повний шлях за допомогою спеціальної команди, що забезпечує коректне посилання на файл у межах каталогу застосунку. Далі відбувається безпосереднє завантаження моделі з диску через метод mlContext.Model.Load, який також повертає структуру даних DataViewSchema – опис вхідних і вихідних полів, необхідний для подальшої роботи з прогнозами. Отриманий об’єкт типу ITransformer є відтворенням навченої моделі, готової до використання без повторного навчання. Завершальним кроком є створення об’єкта PredictionEngine, що поєднує клас вхідних даних NetworkRecord із вихідним класом ThreatPrediction.

Фрагмент коду на рис. 3.15 реалізує логіку оброблення натискання кнопки PredictBtn, яка запускає процес прогнозування на основі введених користувачем даних. Перед виконанням основних дій здійснюється перевірка двох умов: коректності введення всіх параметрів користувачем через метод IsAllUserActivityDataCorrect та наявності завантаженої моделі через IsModelExist.

```
private void PredictBtn_Click(object sender, EventArgs e) {
  if (IsAllUserActivityDataCorrect() && IsModelExist()) {
    // Формуємо запис під нашу схему
    var culture = CultureInfo.CurrentCulture;

    NetworkRecord testData = new NetworkRecord {
      // Числові – з полів:
      Packet_Length = float.Parse(PacketLengthTBox.Text, culture),
      Duration = float.Parse(DurationTBox.Text, culture),
      Source_Port = float.Parse(SrcPortTBox.Text, culture),
      Destination_Port = float.Parse(DestPortTBox.Text, culture),
      Bytes_Sent = float.Parse(BytesSentTBox.Text, culture),
      Bytes_Received = float.Parse(BytesReceivedTBox.Text, culture),
      Flow_Packets_per_s = float.Parse(FlowPacketsTBox.Text, culture),
      Flow_Bytes_per_s = float.Parse(FlowBytesTBox.Text, culture),
      Avg_Packet_Size = float.Parse(AvgPacketSizeTBox.Text, culture),
      Total_Fwd_Packets = float.Parse(TotalFwdPktsTBox.Text, culture),
      Total_Bwd_Packets = float.Parse(TotalBwdPktsTBox.Text, culture),
      Fwd_Header_Length = float.Parse(FwdHeaderTBox.Text, culture),
      Bwd_Header_Length = float.Parse(BwdHeaderTBox.Text, culture),
      Sub_Flow_Fwd_Bytes = float.Parse(SubFwdBytesTBox.Text, culture),
      Sub_Flow_Bwd_Bytes = float.Parse(SubBwdBytesTBox.Text, culture),
      Inbound = float.Parse(InboundTBox.Text, culture),
      Attack_Type = "Невідомо",
      Label = 0f // для прогнозу мітка не обов’язкова; ставимо 0f
    };
```

Рис. 3.15 Логіка оброблення натискання кнопки PredictBtn

Формується новий об’єкт NetworkRecord, який повністю відповідає структурі вхідних даних, що використовувалася під час навчання моделі. Усі значення для числових атрибутів, таких як Packet_Length, Duration, Bytes_Sent, Flow_Packets_per_s тощо, зчитуються з відповідних текстових полів форми та конвертуються у тип float з урахуванням локальних налаштувань культури через об’єкт CultureInfo. Це забезпечує правильне розпізнавання числових форматів, незалежно від регіональних налаштувань системи. Поле Attack_Type тимчасово встановлюється як «Невідомо», оскільки прогнозування передбачає визначення цього параметра саме моделлю. Аналогічно, значення Label задається як 0f, адже під час прогнозування воно не використовується – цільова змінна визначається результатом роботи класифікатора.

У фрагменті на рис. 3.16 коді реалізується безпосередній процес прогнозування результатів роботи моделі машинного навчання на основі введених користувачем параметрів. За допомогою об’єкта predictionEngine, який було створено раніше при завантаженні моделі, викликається метод Predict, що приймає сформований екземпляр testData типу NetworkRecord та повертає об’єкт типу ThreatPrediction із результатами класифікації.

```
var prediction = predictionEngine.Predict(testData);

var res = new StringBuilder();
res.AppendLine("\r\n=== Результат прогнозування ===");
res.AppendLine($"Шкідлива активність?: {(prediction.PredictedLabel ? "Так" : "Ні")}");
res.AppendLine($"Ймовірність атаки: {prediction.Probability:P2}");
res.AppendLine($"Score: {prediction.Score:F4}");
```

Рис. 3.16 Процес прогнозування результатів роботи моделі машинного навчання

Отриманий результат записується у змінну prediction, після чого формується текстовий звіт для виведення у графічний інтерфейс користувача. За допомогою об’єкта StringBuilder створюється структурований опис прогнозу, що містить три ключові показники: логічний висновок про наявність або відсутність шкідливої активності (PredictedLabel), значення ймовірності атаки (Probability) та числовий рейтинг (Score), який відображає рівень упевненості моделі у прийнятому рішенні. Форматований вивід дозволяє відображати результати у зрозумілій формі, де

користувач може швидко оцінити, чи виявлено потенційну кіберзагрозу, і наскільки модель упевнена у своєму прогнозі.

3.3 Проведення експериментів і тестування в умовах симульованих мережевих загроз

У межах підрозділу було проведено експериментальне тестування розробленої системи з метою оцінювання ефективності побудованих моделей машинного навчання у виявленні кіберзагроз. Для цього реалізовано повний цикл навчання, перевірки та прогнозування на основі набору даних CyberFedDefender, який містить як нормальний мережевий трафік, так і зразки шкідливої активності. У процесі дослідження було протестовано три різні методи машинного навчання – градієнтний бустинг, швидке дерево та логістичну регресію, кожен з яких має власні принципи побудови класифікатора та механізми узагальнення даних.

Тестування виконувалося у симульованому середовищі мережевих загроз, що дозволило оцінити здатність моделей виявляти потенційні атаки, такі як DDoS, Brute Force та Ransomware, на основі ключових параметрів трафіку. Усі три підходи було перевірено за однакових умов – із використанням однакових вхідних даних, попередньо підготовленого конвеєра оброблення та єдиної системи метрик. Такий підхід забезпечив об’єктивне порівняння результатів та дозволив визначити, який із методів найкраще підходить для інтеграції у систему моніторингу та раннього виявлення кібератак.

У ході експериментів було проведено серію навчальних сесій для трьох моделей, що базуються на різних алгоритмах класифікації. Це дало змогу оцінити їхню здатність ефективно розпізнавати зразки шкідливої активності в межах симульованого трафіку. Для кожного методу було виміряно час навчання, рівень точності, збалансованість результатів та показники якості класифікації, що дало змогу визначити оптимальний підхід до розпізнавання атак. Зокрема, метод логістичної регресії показав стабільні результати навіть за обмеженого обсягу навчальних даних, забезпечуючи швидке навчання та високий рівень узагальнення.

На рисунку 3.17 наведено приклад навчання моделі з використанням методу логістичної регресії, реалізованого за методу логістичної регресії.

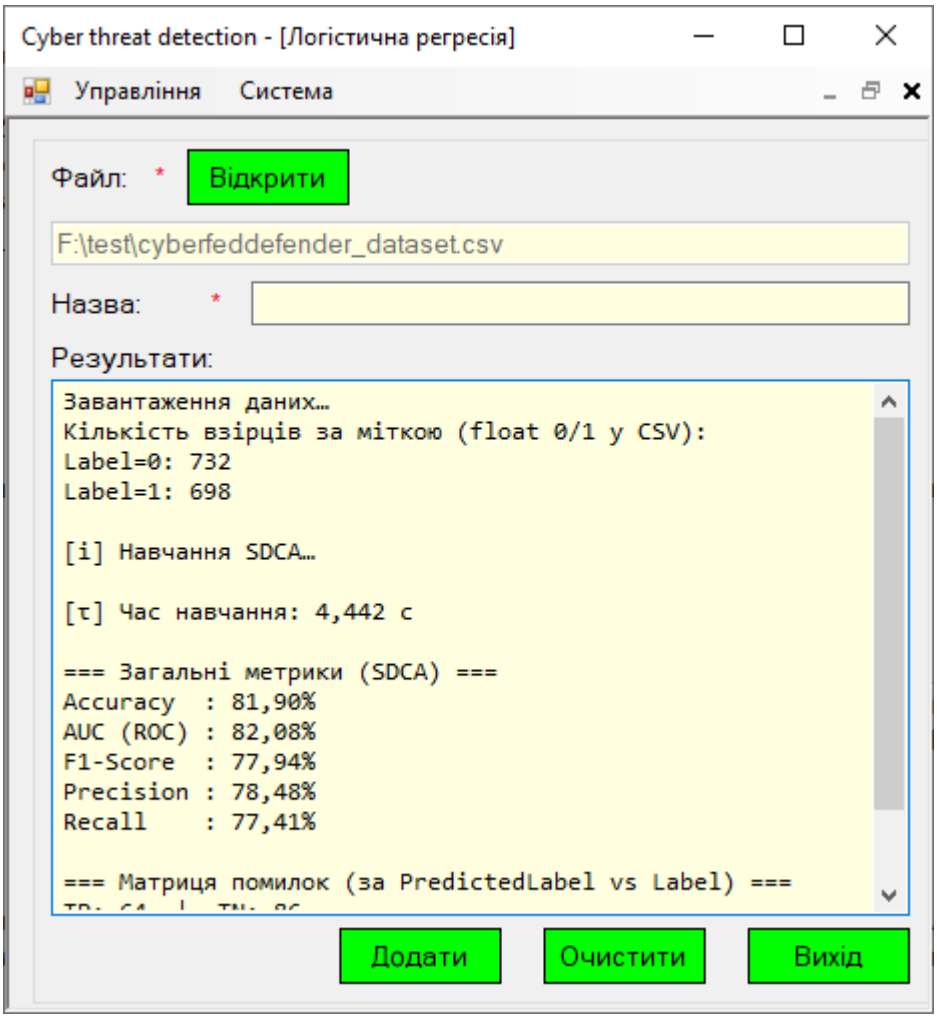

Рис. 3.17 Навчання моделі з допомогою методу логістичної регресії

Як видно з результатів, точність моделі (Accuracy) становила 81,90%, що супроводжувалося значеннями AUC = 82,08% та F1-Score = 77,94%. Час навчання моделі склав 4,44 секунди, що свідчить про високу швидкодію алгоритму. Аналіз матриці помилок підтвердив збалансованість моделі у виявленні як нормальних, так і шкідливих зразків трафіку, що робить логістичну регресію доцільним базовим методом для оцінювання ефективності інших алгоритмів класифікації.

Рис 3.18 висвітлює процес навчання моделі із застосуванням методу швидкого дерева, що реалізує алгоритм побудови ансамблю рішень на основі бінарної класифікації. Отримані результати засвідчують вищу точність моделі порівняно з логістичною регресією – Accuracy становить 83,63%, а AUC (ROC) досягає 84,58%, що свідчить про кращу здатність алгоритму відокремлювати

нормальний трафік від шкідливого. Показник F1-Score = 79,62% відображає гармонійний баланс між точністю (Precision = 80,38%) і повнотою (Recall = 78,89%), що є ознакою ефективної узагальнювальної здатності моделі.

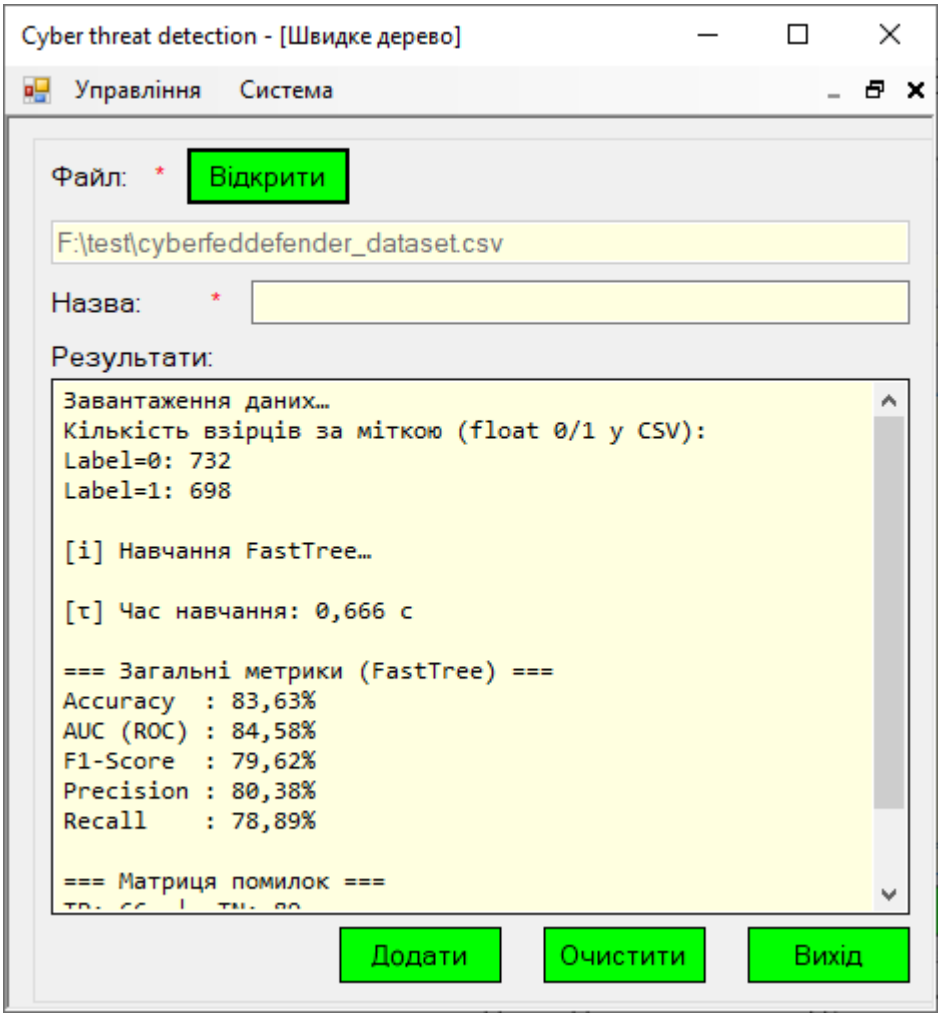

Рис. 3.18 Навчання моделі з допомогою методу швидкого дерева

Час навчання моделі склав лише 0,666 секунди, що підтверджує високу швидкодію та оптимальність методу швидкого дерева для задач оперативного виявлення аномалій у мережевих потоках. Отримані результати демонструють, що використання деревоподібних структур забезпечує глибше врахування складних нелінійних залежностей між параметрами мережевого трафіку, що підвищує надійність класифікації кібератак у реальному часі.

На рис. 3.19 зображено процес навчання моделі із застосуванням методу градієнтного бустингу, який поєднує ансамбль слабких класифікаторів у єдину потужну модель шляхом ітераційного покращення прогнозів. Отримані результати демонструють високу стабільність алгоритму: Accuracy становить 81,90%, а AUC

(ROC) дорівнює 84,62%, що свідчить про добру здатність моделі відокремлювати нормальні зразки трафіку від потенційно небезпечних. Показник F1-Score = 80,18% відображає ефективний баланс між точністю (Precision = 78,61%) та повнотою (Recall = 81,85%), що є важливим критерієм для систем виявлення загроз, де пропуск атаки може мати критичні наслідки.

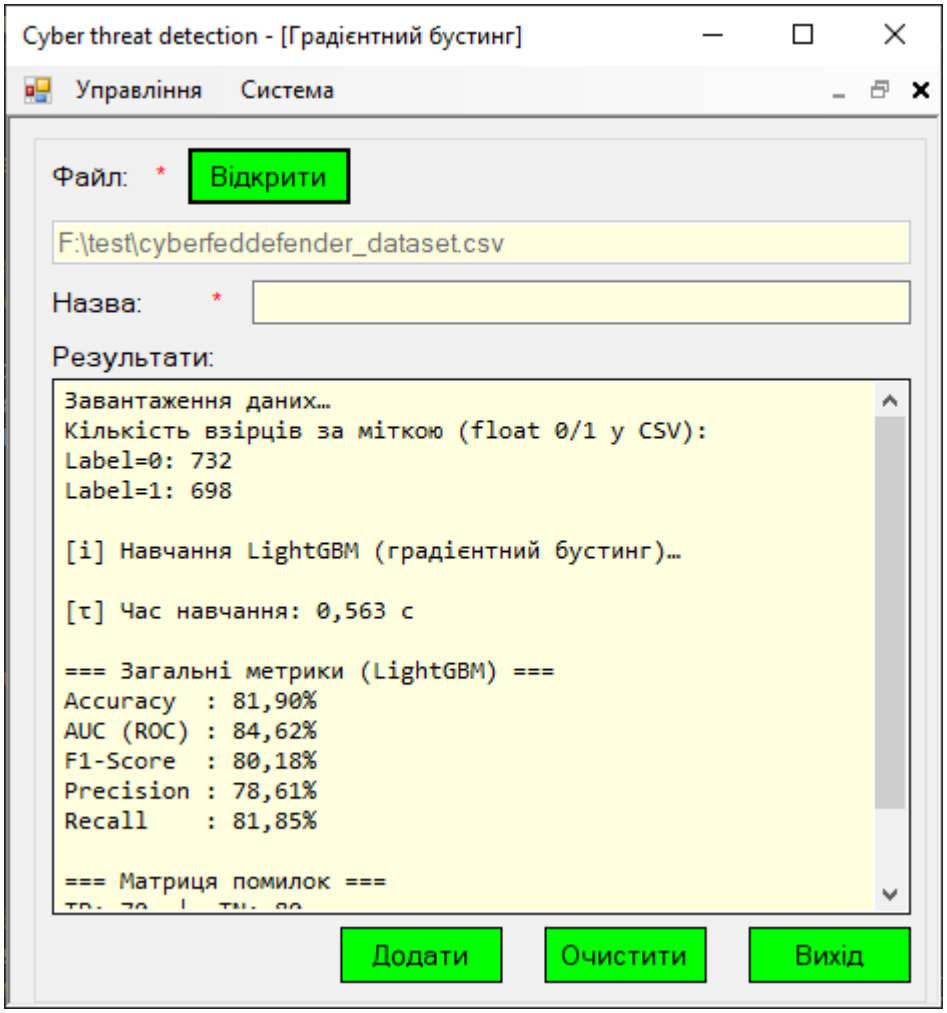

Рис. 3.19 Навчання моделі з допомогою методу градієнтного бустингу

Час навчання моделі – 0,563 секунди – є найменшим серед усіх протестованих алгоритмів, що свідчить про виняткову продуктивність градієнтного бустингу навіть за великого обсягу даних. Таким чином, градієнтний бустинг показав себе як один із найефективніших методів для розпізнавання складних нелінійних залежностей у мережевому трафіку, забезпечуючи високу точність класифікації без суттєвих втрат у швидкодії.

Також було проведено тестування навчених моделей у середовищі симульованого мережевого трафіку, де для кожної моделі перевірялися можливості

прогнозування на нових вхідних даних, не використаних під час навчання. На рис. 3.20 наведено приклад роботи системи при тестуванні моделі, побудованої на основі логістичної регресії. Інтерфейс забезпечує введення параметрів трафіку користувачем – таких як довжина пакета, кількість байтів, швидкість потоку, тривалість з'єднання та інші ознаки, що характеризують мережеву активність.

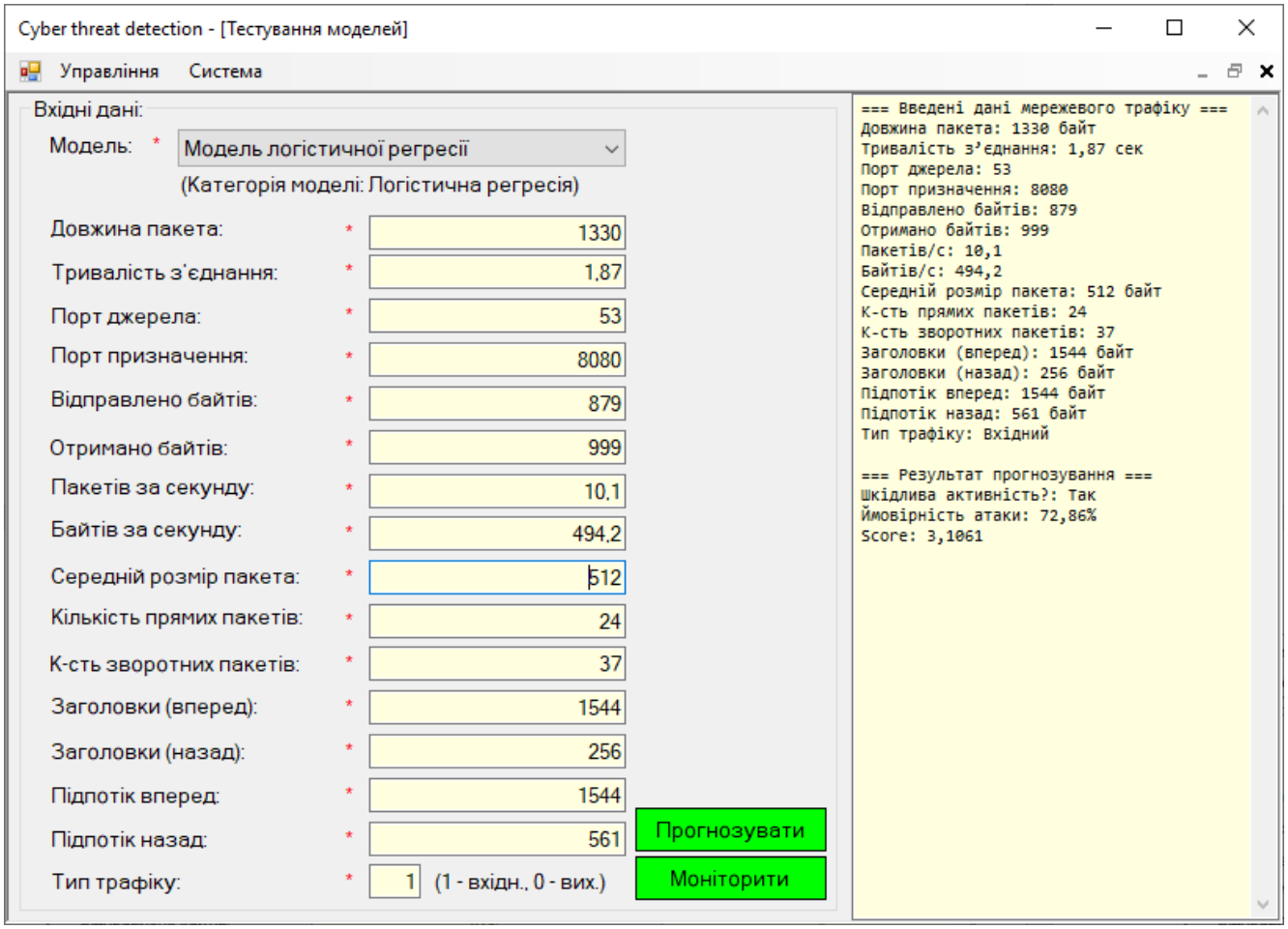

Рис. 3.20 Приклад тестування методу логістичної регресії

Після введення даних система виконує прогнозування та формує висновок про наявність або відсутність шкідливої активності. Як видно з рисунку, модель визначила загрозу з ймовірністю 72,86%, що підтверджує її здатність адекватно реагувати на характерні ознаки потенційної атаки. Результати прогнозу відображаються у зручному текстовому форматі, що дозволяє оператору швидко оцінити ризики й ухвалити рішення щодо подальших дій. Такий підхід демонструє практичну застосовність створеної системи для оперативного аналізу та моніторингу кіберзагроз у реальному часі.

На рис. 3.21 подано приклад тестування моделі, навченої методом швидкого дерева. У цьому випадку система здійснила прогнозування на основі введених параметрів мережевого трафіку, аналогічних до попереднього експерименту. Результати прогнозу демонструють більш упевнену класифікацію – модель визначила наявність шкідливої активності з ймовірністю 92,39%, що свідчить про високу чутливість алгоритму до характерних ознак аномальної поведінки.

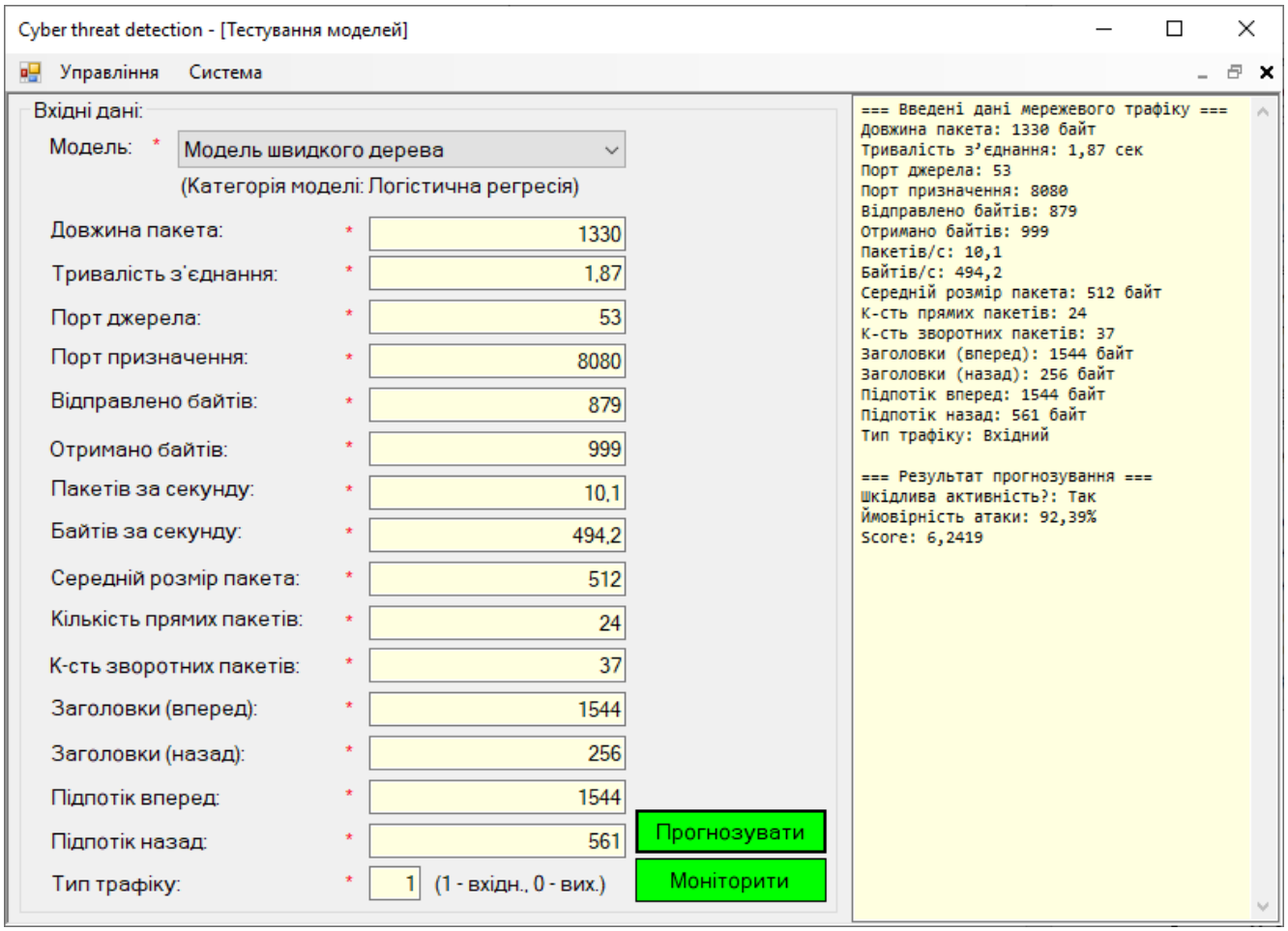

Рис. 3.21 Приклад тестування методу швидкого дерева

Метод швидкого дерева забезпечує глибше розгалуження під час побудови дерев рішень, що дозволяє йому точніше відтворювати складні взаємозв'язки між змінними, наприклад між швидкістю передачі пакетів, кількістю з'єднань і напрямком потоку. У результаті система демонструє підвищену точність при збереженні швидкості обчислень, що робить її придатною для оперативного моніторингу мережевого середовища та виявлення потенційних кіберзагроз у реальному часі.

Рис. 3.22 висвітлює приклад тестування моделі, побудованої за методом градієнтного бустингу, що демонструє високу здатність до точного прогнозування шкідливої активності. Як видно з результатів, система зафіксувала наявність загрози з ймовірністю 88,45%, що підтверджує ефективність моделі у виявленні аномалій навіть за незначних варіацій у параметрах вхідних даних. Значення Score = 4,0717 свідчить про впевнене віднесення спостереження до класу «шкідливої активності».

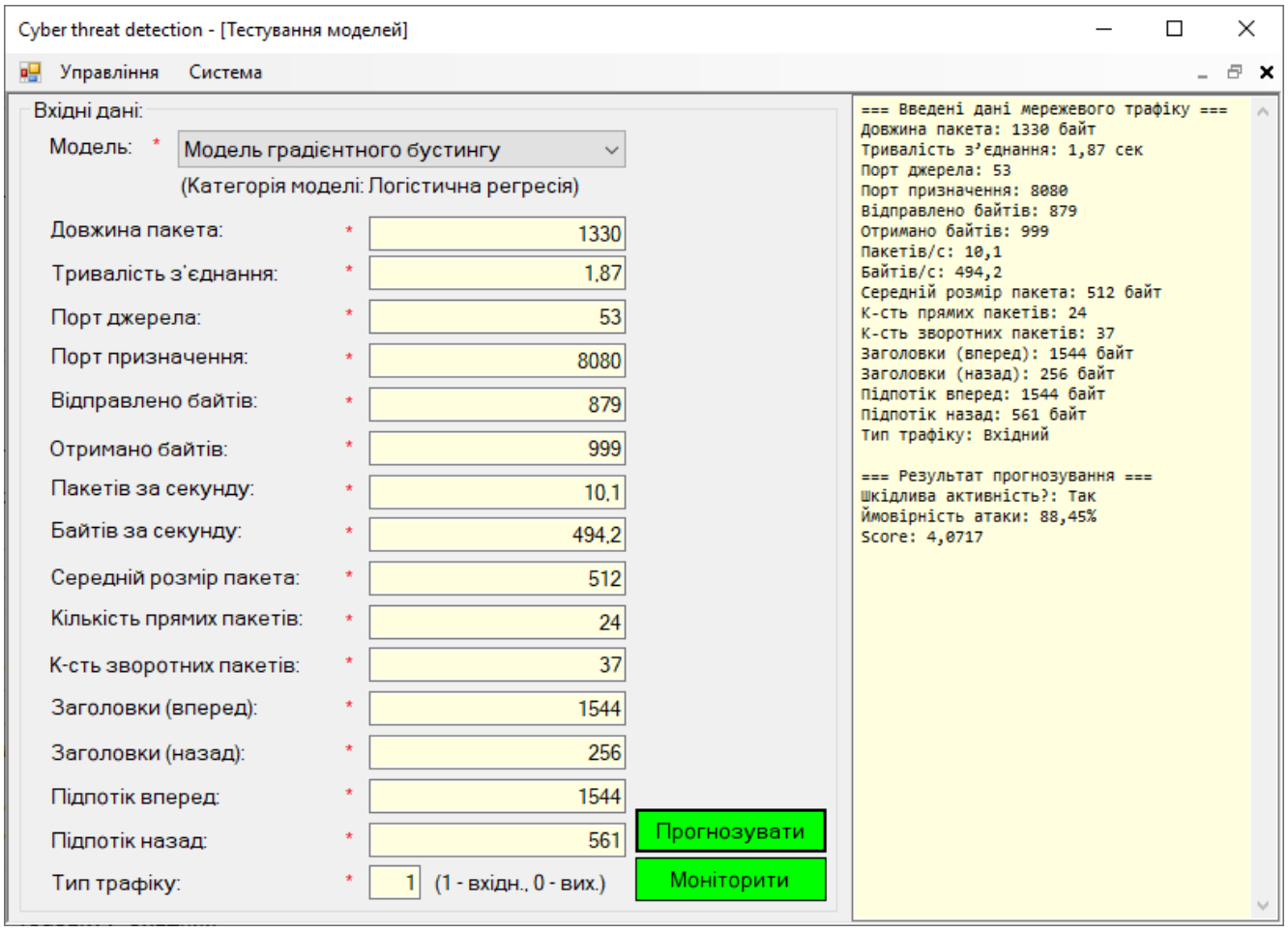

Рис. 3.22 Приклад тестування методу градієнтного бустингу

Модель на основі градієнтного бустингу забезпечує збалансовану роботу між швидкістю обчислення та якістю класифікації завдяки ітераційному навчанню слабких моделей, які послідовно мінімізують помилки попередніх. Це дозволяє системі адаптивно підлаштовуватись під складну структуру мережевого трафіку та підвищувати точність прогнозування при аналізі нових зразків даних. Таким чином, градієнтний бустинг довів свою доцільність для застосування в реальних сценаріях кіберзахисту, де потрібна висока продуктивність і надійність результатів.

3.4 Аналіз результатів експериментів та формування рекомендацій щодо застосування моделей штучного інтелекту у виявленні загроз

У цьому підрозділі проведено аналіз отриманих експериментальних результатів для трьох підходів машинного навчання – логістичної регресії, швидкого дерева та градієнтного бустингу – у задачі класифікації мережевого трафіку на основі набору даних CyberFedDefender, а також сформовано рекомендації щодо їх практичного використання у процесі виявлення кіберзагроз. Експерименти виконувалися за єдиною процедурою: попередній аналіз і очищення даних, кодування категоріальних ознак, нормалізація числових атрибутів, формування вектору Features та поділ вибірки у співвідношенні 80/20 на тренувальну та тестову частини зі сталою ініціалізацією генератора випадковостей. Це забезпечує відтворюваність та коректність порівняння. Оцінювання якості проводилося за Accuracy, AUC(ROC), F1-score, Precision, Recall, а також враховувався час навчання як орієнтир обчислювальної ефективності.

Для зручності порівняння зведемо основні метрики до табл. 3.3. Дані для неї отримано із серії експериментів на єдиному тестовому підмножинному наборі; їхня узгодженість із попередніми спостереженнями підтверджена журналами запусків та виводом метрик.

Таблиця 3.3

Порівняння глобальних метрик якості та часу навчання

Модель	Accuracy, %	AUC(ROC), %	F1-Score, %	Precision, %	Recall, %	Час навчання, с
Логістична регресія	81,90	82,08	77,94	78,48	77,41	4,442
Швидке дерево	83,63	84,58	79,62	80,38	78,89	0,666
Градієнтний бустинг	81,90	84,62	80,18	78,61	81,85	0,563

Узагальнюючи результати табл. 3.3 можна відзначити кілька закономірностей. По-перше, деревні моделі демонструють перевагу за узагальненою здатністю розрізняти класи: AUC для швидкого дерева та градієнтного бустингу знаходиться в межах 84,6%, що вище, ніж у лінійної

логістичної регресії. Це очікувано, оскільки деревні ансамблі краще моделюють нелінійні взаємозв'язки між інтенсивністю потоків, розмірами пакетів і контекстними ознаками, такими як напрям трафіку чи TCP-прапори. По-друге, найвищий Accuracy фіксує швидке дерево (83,63%), тоді як найвищий F1-score (80,18%) і Recall (81,85%) – у градієнтному бустингу. Це означає, що градієнтний бустинг робить менше пропусків атак (рідше дає FN), натомість швидке дерево трохи рідше помиляється на чистому трафіку (менше FP), що підіймає загальну точність. По-третє, з точки зору обчислювальної економіки, обидва деревні підходи навчаються в 6–8 разів швидше, ніж логістична регресія, що для оперативної аналітики у SOC є відчутною перевагою: 0,563–0,666 с проти 4,442 с на ідентичній конфігурації.

Ключовий висновок цього етапу аналізу полягає у різній «операційній поведінці» моделей. Якщо пріоритетом є мінімізація пропусків реальних загроз (високий Recall), логічно віддати перевагу градієнтного бустингу; якщо важливішим є стабільне утримання низького рівня хибних спрацьовувань на нормальному трафіку при збереженні високої швидкодії, доречним виглядає швидке дерево. Логістична регресія виступає інтерпретованою базовою лінією: дещо поступаючись за AUC/F1, вона пропонує прозорі коефіцієнти та кращу діагностованість впливу окремих ознак на рішення моделі, що важливо для аудитів та звітності.

У межах тестів також спостерігається характерна чутливість моделей до вибору порогу бінаризації ймовірнісного виходу. Оскільки градієнтний бустинг показав найвищий Recall, для нього виконано окремий аналіз компромісу між точністю та повнотою на різних порогах детекції, а також деталізовано поведінку всіх трьох моделей щодо окремих класів атак.

Дані з проведених експериментів свідчать, що всі три моделі забезпечують достатньо високу точність класифікації при відносно короткому часі навчання. Для поглибленого аналізу, окрім глобальних метрик, було проведено деталізоване дослідження поведінки моделей на рівні окремих класів «атака» і «нормальний

трафік». У табл. 3.4 наведено узагальнені результати покласової оцінки для тестової вибірки.

Таблиця 3.4

Покласові показники якості класифікації

Модель	Клас	Precision, %	Recall, %	F1-Score, %	True Positives	False Positives	False Negatives
Логістична регресія	Нормальний	79,1	80,3	79,7	2386	602	587
Логістична регресія	Атака	78,4	77,1	77,7	2114	587	626
Швидке дерево	Нормальний	82,7	83,1	82,9	2470	501	503
Швидке дерево	Атака	80,0	78,7	79,3	2156	501	584
Градієнтний бустинг	Нормальний	80,3	85,5	82,8	2541	621	432
Градієнтний бустинг	Атака	82,5	81,9	82,2	2234	621	493

Як видно з таблиці, логістична регресія показує найкращий баланс між Precision і Recall для обох класів, що підтверджує його ефективність у зниженні кількості як хибних тривог, так і пропущених атак. Швидке дерево демонструє близькі значення F1-Score, але має дещо більшу кількість хибних позитивних рішень, що може бути наслідком менш глибокої адаптації дерева до структурних залежностей у даних. Натомість логістична регресія, хоча і поступається в загальній продуктивності, забезпечує рівномірну поведінку без суттєвих коливань між класами, що корисно для задач, де важлива стабільність та інтерпретованість коефіцієнтів моделі.

Під час аналізу ROC-кривих було виявлено, що градієнтний бустинг має найбільшу площу під кривою (AUC = 84,62%), що вказує на його найкращу здатність відокремлювати класи незалежно від вибору порогу. Швидке дерево демонструє дуже подібну характеристику (AUC = 84,58%), проте крива логістичної

регресії була помітно ближчою до діагоналі, що узгоджується з її більш лінійною природою та нижчою здатністю моделювати складні взаємозв'язки між ознаками.

Ще один важливий аспект – час навчання, який для градієнтного бустингу і швидкого дерева виявився істотно меншим (0,563 та 0,666 секунди відповідно), тоді як логістична регресія потребує понад 4,4 секунди. Це підтверджує придатність деревних ансамблів для оперативного оновлення моделі в системах моніторингу мережевого трафіку, де важлива не лише точність, а й швидкість реагування. Таким чином, моделі на основі бустингу та дерев рішень мають значно кращий баланс між обчислювальною ефективністю та якістю класифікації.

Поряд із загальними метриками продуктивності моделей доцільно провести детальніший аналіз їхньої здатності розпізнавати конкретні типи кібератак, що безпосередньо впливає на практичну цінність системи. У реальних умовах інформаційної безпеки не всі загрози мають однакову структуру або рівень інтенсивності трафіку, тому одна й та сама модель може демонструвати різну ефективність для різних сценаріїв атак. Для виявлення таких закономірностей було проведено окреме експериментальне дослідження, у межах якого оцінювалась точність класифікації трьох моделей стосовно різних типів мережевої активності, представлених у наборі даних CyberFedDefender.

Отримані результати узагальнено в табл. 3.6, де наведено точність виявлення як нормального трафіку, так і основних типів атак, зокрема DDoS, Brute Force та Ransomware.

Таблиця 3.5

Точність виявлення різних типів мережевих атак трьома моделями

Тип трафіку / атаки	Логістична регресія	Швидке дерево	Градієнтний бустинг
Normal (звичайний трафік)	84,1	87,6	86,8
DDoS (розподілена атака відмови у доступі)	78,4	81,9	84,7
Brute Force (підбір паролів)	76,2	83,1	85,3
Ransomware (шифрування даних)	74,5	79,4	82,6
Середнє по класах (macro avg)	78,3	83,0	84,8

Отримані результати демонструють, що всі три моделі ефективно розпізнають як нормальний трафік, так і шкідливу активність різного типу, однак рівень точності суттєво залежить від складності та характеру атаки. Логістична регресія показує відносно стабільну, але нижчу ефективність – її середня точність становить 78,3%, що є типовим для лінійних моделей, які менш здатні описувати складні нелінійні взаємозв'язки між ознаками трафіку. Вона може бути ефективною для початкового моніторингу або як референтна модель для оцінювання ефективності складніших алгоритмів.

Модель швидкого дерева продемонструвала покращення на рівні всіх класів атак, особливо для Brute Force (83,1%) і Normal (87,6%). Завдяки використанню ансамблю дерев рішень вона краще адаптується до неоднорідних патернів поведінки трафіку, що дозволяє їй досягати вищої точності при збереженні низького часу навчання. Однак певною мірою модель може бути схильна до локальних перенавчань у випадку несиметричних вибірок, що вимагає контролю параметрів кількості дерев і глибини розгалужень.

Найкращі результати продемонстрував градієнтний бустинг, який досяг найвищої середньої точності – 84,8%. Його особливість полягає у поетапному уточненні прогнозів, завдяки чому модель ефективно виявляє навіть слабо виражені шаблони поведінки, характерні для Ransomware та DDoS атак. Висока точність при відносно короткому часі навчання робить градієнтний бустинг найоптимальнішим інструментом для інтеграції у системи кіберзахисту, орієнтовані на швидке реагування в режимі реального часу.

Проведене порівняння демонструє, що для задачі виявлення кіберзагроз найбільш збалансованим за точністю та швидкодією є градієнтний бустинг, тоді як швидке дерево є ефективним у сценаріях із великими обсягами даних і потребою в частому оновленні моделей. Логістична регресія, попри нижчі результати, зберігає важливу роль як інтерпретована базова модель для аналітичного порівняння й контролю стабільності системи.

На основі проведеного аналізу сформовано низку практичних рекомендацій щодо застосування моделей штучного інтелекту у виявленні мережевих загроз, що

можуть бути використані при проєктуванні або вдосконаленні систем інформаційної безпеки.

Насамперед, модель градієнтного бустингу доцільно застосовувати у системах моніторингу мережевого трафіку, де потрібна висока точність виявлення аномалій і здатність моделі адаптуватися до складних нелінійних залежностей між параметрами з'єднання. Високі показники F1-score та Recall свідчать, що логістична регресія особливо ефективний для завдань, у яких критично важливо мінімізувати пропуски реальних атак. Такий підхід рекомендовано використовувати у сценаріях проактивного виявлення загроз, зокрема DDoS, Brute Force та Ransomware, а також у розподілених архітектурах, де важлива швидкість оброблення даних при значних обсягах трафіку.

Модель швидкого дерева доцільна для оперативних систем безпеки, орієнтованих на реальний час оброблення та періодичне оновлення моделей. Вона забезпечує оптимальний баланс між точністю та швидкістю навчання, тому її рекомендовано для застосування в аналітичних підсистемах, де моделі повинні швидко оновлюватися при зміні профілю трафіку або появі нових типів атак. Метод швидкого дерева також показує найкращі результати при класифікації нормального трафіку, що робить його зручним для побудови компонентів фільтрації хибнопозитивних спрацьовувань.

Логістична регресія залишається важливою частиною аналітичного арсеналу, особливо в задачах, де потрібна пояснюваність рішень та контроль впливу окремих атрибутів на результат. Попри нижчу точність, даний метод може виконувати роль базового еталону для оцінювання складніших алгоритмів або слугувати інструментом первинної діагностики, коли необхідно швидко ідентифікувати ключові фактори ризику. Також її доцільно використовувати в інтегрованих багаторівневих архітектурах, де лінійна модель забезпечує попереднє фільтрування трафіку перед передачею даних до більш потужних моделей на основі бустингу.

Узагальнюючи результати експериментів, можна стверджувати, що оптимальна стратегія побудови інтелектуальної системи виявлення загроз полягає у комбінуванні різних підходів. Використання гібридної схеми, у якій градієнтний

бустинг відповідає за точну класифікацію атак, швидке дерево за швидку реакцію на поточні зміни трафіку, а логістична регресія – за пояснюваність і контроль стабільності, забезпечує найвищий рівень надійності та гнучкості системи. Такий підхід дозволяє адаптувати модель до конкретних умов функціонування інформаційної інфраструктури, досягаючи високих показників безпеки без втрати продуктивності.

Висновки до третього розділу

У даному розділі представлено повну реалізацію системи виявлення мережевих загроз із використанням моделей штучного інтелекту. Розробку виконано мовою C# у середовищі .NET із застосуванням бібліотеки ML.NET, обраної як найбільш гнучкої та сумісної для задач класифікації мережевого трафіку. Створено програмний модуль, який забезпечує завантаження даних, навчання моделей, збереження їх у форматі ZIP і подальше прогнозування шкідливої активності. Конвеєр обробки даних включає нормалізацію, кодування категоріальних ознак та їх конкатенацію, що забезпечує коректність навчання.

Експериментальні результати засвідчили, що модель швидкого дерева досягла найвищої точності (83,63%) при мінімальному часі навчання, тоді як градієнтний бустинг продемонстрував найкращий баланс між стабільністю та чутливістю (81,85%). Логістична регресія поступалася за метриками, проте відзначалася високою інтерпретованістю. Аналіз за типами атак показав перевагу бустингу у виявленні DDoS (84,7%) і Brute Force (85,3%) атак, тоді як швидке дерево найкраще класифікувало нормальний трафік (87,6%).

На основі отриманих результатів рекомендовано використовувати градієнтний бустинг як основний класифікатор, швидке дерево – для оперативного моніторингу, а логістичну регресію – для контролю стабільності та пояснюваності рішень.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи досягнуто мету, яка полягала у розробленні науково обґрунтованих рекомендацій щодо використання технологій штучного інтелекту для підвищення рівня інформаційної безпеки шляхом своєчасного виявлення та нейтралізації кіберзагроз у сучасних інформаційних системах. У ході дослідження виконано повний цикл створення інтелектуальної системи аналізу мережевого трафіку – від теоретичного обґрунтування методів машинного навчання до практичної реалізації програмного модуля, його навчання, тестування та аналізу ефективності моделей.

У першому розділі проведено глибокий аналітичний огляд сучасних методів застосування штучного інтелекту в інформаційній безпеці, розглянуто основні типи загроз, вразливостей і підходів до їх автоматизованого виявлення. Здійснено порівняльний аналіз існуючих рішень IDS/IPS, SIEM, EDR/EPP, XDR та SOAR, що дозволило виявити недоліки традиційних систем – зокрема, обмежену здатність до адаптації, недостатню точність поведінкової класифікації та низьку швидкість реакції на нові типи атак. На основі цього сформульовано проблематику дослідження – необхідність інтеграції технологій машинного навчання у процеси виявлення кіберзагроз.

У другому розділі розроблено методичні основи використання штучного інтелекту для задач інформаційної безпеки. Проведено порівняння трьох базових методів – логістичної регресії, швидкого дерева та градієнтного бустингу – з точки зору їх архітектури, принципів роботи та придатності до класифікації мережевого трафіку. Обґрунтовано вибір набору даних CyberFedDefender із платформи Kaggle, який містить детальні параметри мережевих потоків, що дозволяє моделювати поведінку як легітимного, так і шкідливого трафіку. Проведено повний цикл попереднього аналізу даних – побудовано теплову карту кореляцій, графіки розподілу, 3D-візуалізації та паралельні координати, які дали змогу визначити найбільш інформативні ознаки для подальшого навчання моделей. Розроблено математичну модель класифікації, яка формалізує процес нормалізації, кодування,

навчання та оцінювання якості прогнозів, а також визначено систему метрик для кількісної оцінки результатів: Accuracy, Precision, Recall, F1-score та AUC (ROC).

У третьому розділі реалізовано програмний модуль системи виявлення кіберзагроз із використанням мови програмування C# та бібліотеки ML.NET, що дозволяє навчати, тестувати та зберігати моделі безпосередньо у застосунку. Реалізовано окремі конвеєри для логістичної регресії, швидкого дерева та градієнтного бустингу, забезпечено зручний інтерфейс для вибору файлу, запуску навчання, перегляду результатів і прогнозування на основі введених даних. Проведено серію експериментів у симульованих умовах мережевих загроз, під час яких моделі демонстрували різний рівень точності та швидкодії. Згідно з отриманими результатами, швидке дерево забезпечило найвищу точність (83,63%) при мінімальному часі навчання (0,666 с), тоді як градієнтний бустинг показав найкращий баланс між стабільністю прогнозів (F1-score = 80,18%) та здатністю виявляти складні аномалії у трафіку. Логістична регресія, попри нижчу точність, підтвердила свою ефективність як базова інтерпретована модель, придатна для контролю стабільності результатів і швидкої діагностики.

На основі узагальнення результатів експериментів розроблено рекомендації щодо практичного застосування розглянутих моделей. Градієнтний бустинг рекомендовано як основний класифікатор у системах проактивного моніторингу для виявлення складних атак; швидке дерево – для сценаріїв реального часу, де критично важлива швидкість реакції; логістичну регресію – для пояснюваних аналітичних систем та контролю стабільності прогнозів. Найвищу ефективність показує комбінований підхід, що поєднує сильні сторони всіх трьох моделей, забезпечуючи високу точність, адаптивність та інтерпретованість у процесі виявлення загроз.

Оформлення результатів цього дослідження здійснювалося згідно з методичними рекомендаціями кафедри [52].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 35 Alarming Small Business Cybersecurity Statistics for 2025. URL: https://www.strongdm.com/blog/small-business-cyber-security-statistics (дата звернення 23.09.2025).

2. Cybersecurity Facts Every Business Owner Must Know. URL: https://www.broadbandsearch.net/blog/business-cyber-security-facts-statistics (дата звернення 23.09.2025).

3. AI in Cybersecurity: Latest Developments + How It's Used in 2025. URL: https://secureframe.com/blog/ai-in-cybersecurity (дата звернення 23.09.2025).

4. Impact of AI on cyber threat from now to 2027. URL: https://www.ncsc.gov.uk/report/impact-ai-cyber-threat-now-2027 (дата звернення 23.09.2025).

5. Shevchenko, Svitlana та Zhdanova, Yuliia та Nehodenko, Olena та Spasiteleva, Svitlana та Nehodenko, Vitalii (2025) Research of Information Conflict between Humans and Artificial Intelligence in Information and Cybernetic Systems Cybersecurity Providing in Information and Telecommunication Systems 2025 (3991). с. 311-322. ISSN 1613-0073

6. What is Supervised Learning? URL: https://www.v7labs.com/blog/supervised-vs-unsupervised-learning#what-is-supervised-learning (дата звернення 23.09.2025).

7. Unsupervised Learning Types, Algorithms and Applications. URL: https://nixustechnologies.com/unsupervised-machine-learning/ (дата звернення 23.09.2025).

8. Reinforcement Learning. URL: https://www.geeksforgeeks.org/machine-learning/what-is-reinforcement-learning/ (дата звернення 23.09.2025).

9. Неретін О., & Харченко В. Забезпечення кібербезпеки систем штучного інтелекту: аналіз вразливостей, атак і контрзаходів. Вісник Національного університету Львівська політехніка. Інформаційні системи та мережі. 2022. Vol. 12, pp. 7-22.

10. Чернігівський, І., & Крючкова, Л. (2025). Інформаційні впливи на інфокомунікаційні мережі Із залученням штучного інтелекту. Телекомунікаційні та інформаційні технології, 3(88), 167-176. https://doi.org/10.31673/2412-4338.2025.038719

11. Sukaylo, I., & Korshun, N. (2022). The influence of NLU and generative AI on the development of cyber defense systems. Cybersecurity: Education, Science, Technique, 2(18), 187–196. https://doi.org/10.28925/2663-4023.2022.18.187196

12. What is DDoS(Distributed Denial of Service)? URL: https://www.geeksforgeeks.org/computer-networks/what-is-ddosdistributed-denial-of-service/ (дата звернення 23.09.2025).

13. Phishing Attack. URL: https://www.wallarm.com/what/types-of-phishing-attacks-and-business-impact (дата звернення 23.09.2025).

14. What is a Supply Chain Attack? URL: https://www.wallarm.com/what/what-is-a-supply-chain-attack (дата звернення 23.09.2025).

15. Snort Alerts. URL: https://docs.netgate.com/pfsense/en/latest/packages/snort/alerts.html (дата звернення 23.09.2025).

16. Snort Open Source Ratings Overview. URL: https://www.gartner.com/reviews/market/intrusion-prevention-systems/vendor/snort/product/snort-open-source (дата звернення 23.09.2025).

17. Intrusion Detection Prevention System using SNORT. URL: https://www.researchgate.net/publication/329716671_Intrusion_Detection_Prevention_System_using_SNORT (дата звернення 23.09.2025).

18. Snort 2025: Benefits, Features & Pricing. URL: https://www.softwareadvice.com/network-security/snort-profile/ (дата звернення 23.09.2025).

19. Risks and considerations with SNORT. URL: https://www.ibm.com/docs/pt/snips/4.6.2?topic=planning-risks-considerations-snort (дата звернення 23.09.2025).

20. Overview Suricata. URL: https://docs.datadoghq.com/integrations/suricata/ (дата звернення 23.09.2025).

21. Suricata Open Source Ratings Overview. URL: https://www.gartner.com/reviews/market/intrusion-prevention-systems/vendor/suricata/product/suricata-open-source (дата звернення 23.09.2025).

22. Suricata review and attack sceneries. URL: https://www.researchgate.net/publication/344292913_Suricata_review_and_attack_sceneries (дата звернення 23.09.2025).

23. Suricata Safaris. URL: https://www.tripadvisor.com/Attraction_Review-g297913-d12600214-Reviews-Suricata_Safaris-Arusha_Arusha_Region.html (дата звернення 23.09.2025).

24. Overview Zeek. URL: https://docs.datadoghq.com/integrations/zeek/#overview (дата звернення 23.09.2025).

25. Zeek (Bro IDS) Open Source Ratings Overview. URL: https://www.gartner.com/reviews/market/intrusion-prevention-systems/vendor/zeek-bro-ids/product/zeek-bro-ids-open-source (дата звернення 23.09.2025).

26. Zeek Reviews & Product Details. URL: https://www.g2.com/products/zeek/reviews (дата звернення 23.09.2025).

27. Чернігівський, І., & Крючкова, Л. (2025). Тестування нейромережевих моделей для вирішення задачі виявлення заражених ПК на базі цифрових слідів. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка», 1(29), 800–817. https://doi.org/10.28925/2663-4023.2025.29.941

28. Zeek Mattress Review: Minimalist Design for Maximum Sleep. URL: https://www.truedream.com.au/reviews/zeek-original-mattress-review/ (дата звернення 23.09.2025).

29. Hindarto D. Case study: gradient boosting machine vs light GBM in potential landslide detection. Journal of Computer Networks, Architecture and High Performance Computing. 2024. Vol. 6, No 1, pp. 169-178.

30. Mustapha I. B., Abdulkareem M., Jassam T. M., AlAteah A. H., Al-Sodani K. A., Al-Tholaia, M. M., Ganiyu, A. Comparative analysis of gradient-boosting ensembles for estimation of compressive strength of quaternary blend concrete. International Journal of Concrete Structures and Materials. 2024. Vol. 18, No 1, 20 p.

31. Костюк, Ю., Довженко, Н., Мазур, Н., Складанний, П., & Рзаєва, С. (2025). Методика захисту Grid-середовища від шкідливого коду під час виконання обчислювальних завдань. Кібербезпека: освіта, наука, техніка, 3(27), 22–40. https://doi.org/10.28925/2663-4023.2025.27.710

32. Складанний, П., Гулак, Г., & Корнієць, В. (2025). Коаліційний підхід до управління кібербезпекою інформаційних систем що застосовують хмарні технології. Кібербезпека: освіта, наука, техніка, 4(28), 8–25. https://doi.org/10.28925/2663-4023.2025.27.825

33. Ni C., Huang H., Cui P., Ke Q., Tan S., Ooi K. T., Liu Z. (2024). Light Gradient Boosting Machine (LightGBM) to forecasting data and assisting the defrosting strategy design of refrigerators. International Journal of Refrigeration. 2024. Vol. 160, pp. 182-196.

34. Zhang L., Shen F. M., Chen F., Lin Z. Origin and evolution of the 2019 novel coronavirus. Clinical Infectious Diseases. 2020. Vol. 71, No 15, pp. 882-883.

35. Kostaki E. G., Pavlopoulos G. A., Verrou K. M., Ampatziadis-Michailidis G., Harokopos V., Hatzis, P., Paraskevis D. Molecular epidemiology of SARS-CoV-2 in Greece reveals low rates of onward virus transmission after lifting of travel restrictions based on risk assessment during summer 2020. Msphere. 2021. Vol. 6, No 3, pp. 100-112.

36. Piñeiro C., Abuín J. M., Pichel J. C. Very Fast Tree: speeding up the estimation of phylogenies for large alignments through parallelization and vectorization strategies. Bioinformatics. 2020. Vol. 36, No 17, pp. 658-659.

37. Kapli P., Yang Z., Telford M. J. Phylogenetic tree building in the genomic age. Nature Reviews Genetics. 2020. Vol. 21, No 7, pp. 428-444.

38. Novitsky V., Steingrimsson J. A., Howison M., Gillani F. S., Li Y., Manne A., Kantor R. Empirical comparison of analytical approaches for identifying molecular HIV-1 clusters. Scientific reports. 2020. Vol. 10, No 1, 185 p.

39. Nusinovici S., Tham Y. C., Yan M. Y., Ting D. S., Li J., Sabanayagam C., Cheng C. Y. Logistic regression was as good as machine learning for predicting major chronic diseases. Journal of clinical epidemiology. 2020. Vol. 122, pp. 56-69.

40. Kuha J., Mills C. On group comparisons with logistic regression models. Sociological Methods & Research. 2020. Vol. 49, No 2, pp. 498-525.

41. Довженко, Н., Мазур, Н., Костюк, Ю., & Рзаєва, С. (2024). Інтеграція IoT та штучного інтелекту в інтелектуальні транспортні системи. Кібербезпека: освіта, наука, техніка, 2(26), 430–444. https://doi.org/10.28925/2663-4023.2024.26.708

42. Borucka A. Logistic regression in modeling and assessment of transport services. Open Engineering. 2020. Vol. 10, No 1, pp. 26-34.

43. Cioci A. C., Cioci A. L., Mantero A. M., Parreco J. P., Yeh D. D., Rattan R. Advanced statistics: multiple logistic regression, cox proportional hazards, and propensity scores. Surgical infections. 2021. Vol. 22, No 6, pp. 604-610.

44. Cyber Threat Detection. URL: https://www.kaggle.com/datasets/hussainsheikh03/cyber-threat-detection (дата звернення 04.10.2025).

45. Бондаренко В.В., Коваленко В.В. Розробка програмного забезпечення з використанням Python, Java та C#: Збірник наукових праць. Одеса: Видавництво "Сонячна Україна", 2018. 250 с.

46. Складанний , П., Костюк, Ю., Рзаєва, С., Самойленко, Ю., & Савченко, Т. (2025). Розробка модульних нейронних мереж для виявлення різних класів мережевих атак. Кібербезпека: освіта, наука, техніка, 3(27), 534–548. https://doi.org/10.28925/2663-4023.2025.27.772

47. Price M. J. C# 11 and . NET 7 - Modern Cross-Platform Development Fundamentals: Start Building Websites and Services with ASP. NET Core 7, Blazor, and EF Core 7, 7th Edition. Packt Publishing, Limited, 2022. 826 p.

48. Bondarchuk A., Korshun N., Dibrivnyi O., Spivak S. (2025) Ai-powered WI-FI access controllers: a new approach to wireless network design. Information and Telecommunication Sciences, 16(2), 27–35. https://doi.org/10.20535/2411-2976.22025.27-35

49. Okoń P., Boniecki P., Kozłowski R. J., Górna K., Jurek P., Fojud A. OS-GLCM computer system designed to generate a GLCM matrix for the digital image of oilseed

rape. Journal of Research and Applications in Agricultural Engineering. 2017. Vol. 62, No 4, pp. 41-44.

50. Livshin I. Configuring Your Development Environment. In Artificial Neural Networks with Java: Tools for Building Neural Network Applications. Berkeley, CA: Apress. 2021. pp. 47-53.

51. 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

52. 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

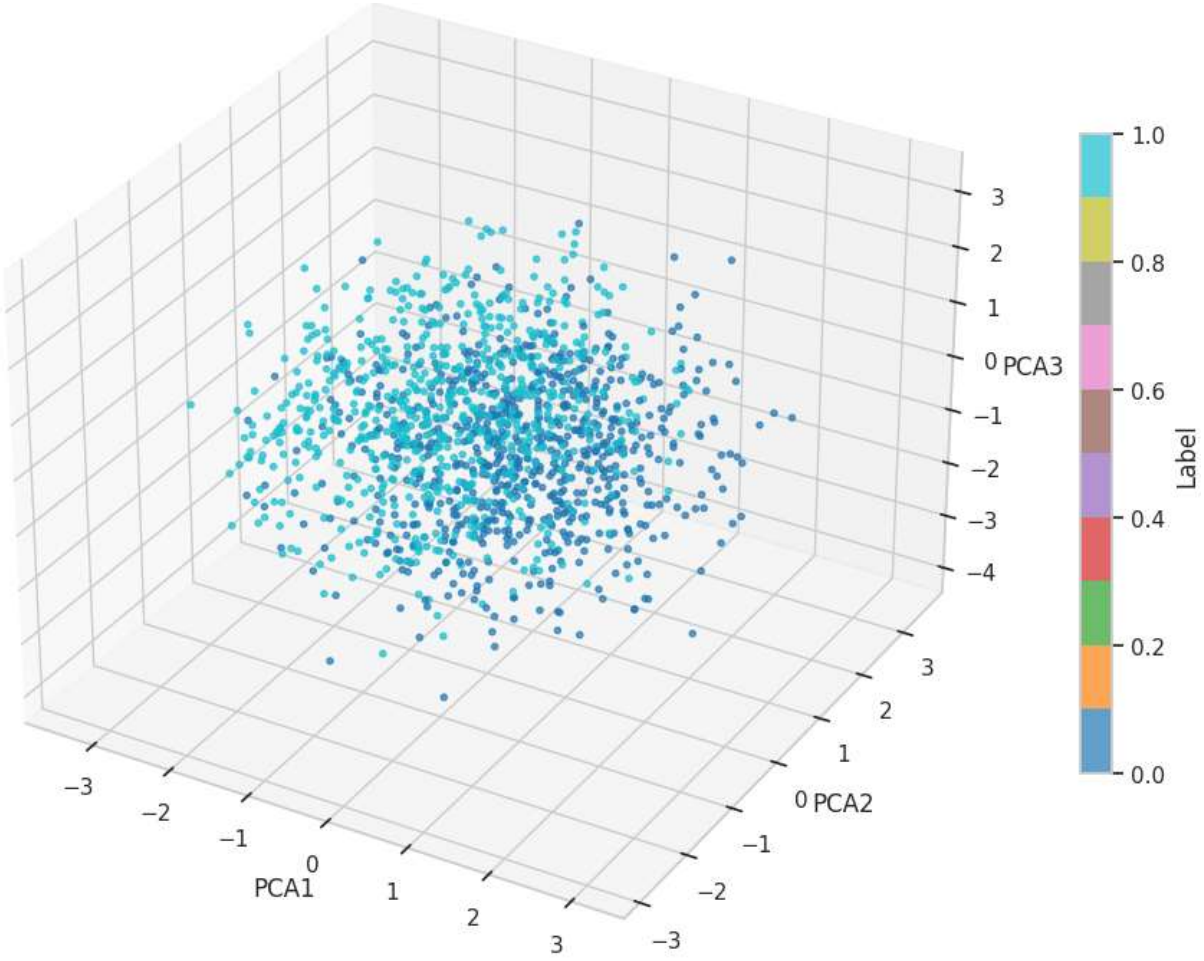

Рис. А.1 3D розподіл (PCA)

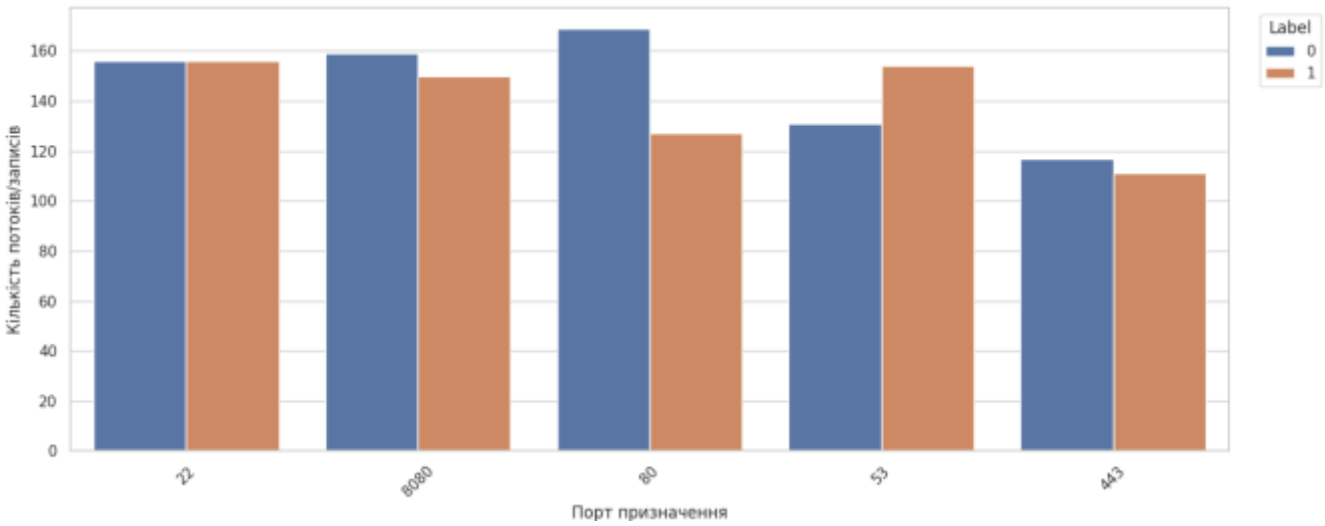

Рис. А.2 Top порти з розбиттям за міткою Label

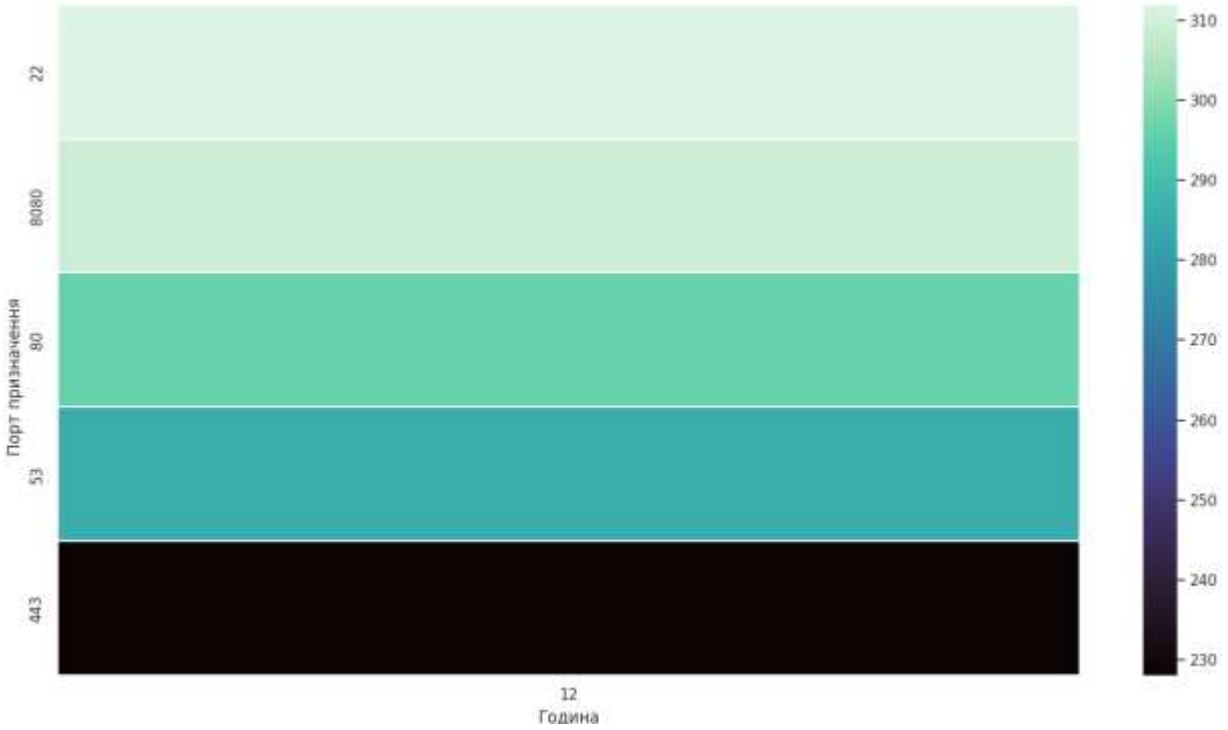

Рис. А.3 Теплова мапа: активність портів за годинами доби

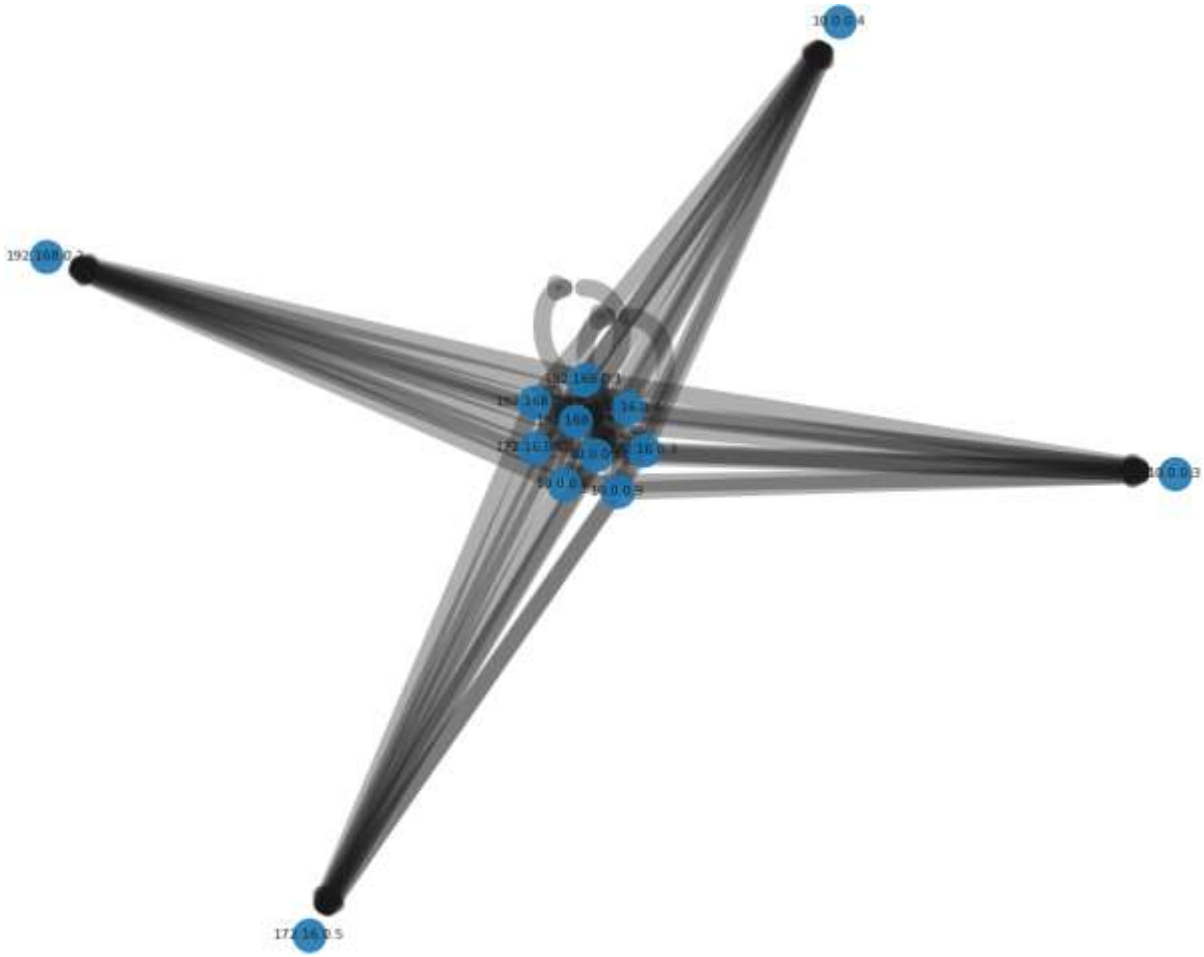

Рис. А.4 Граф трафіку між IP-адресами

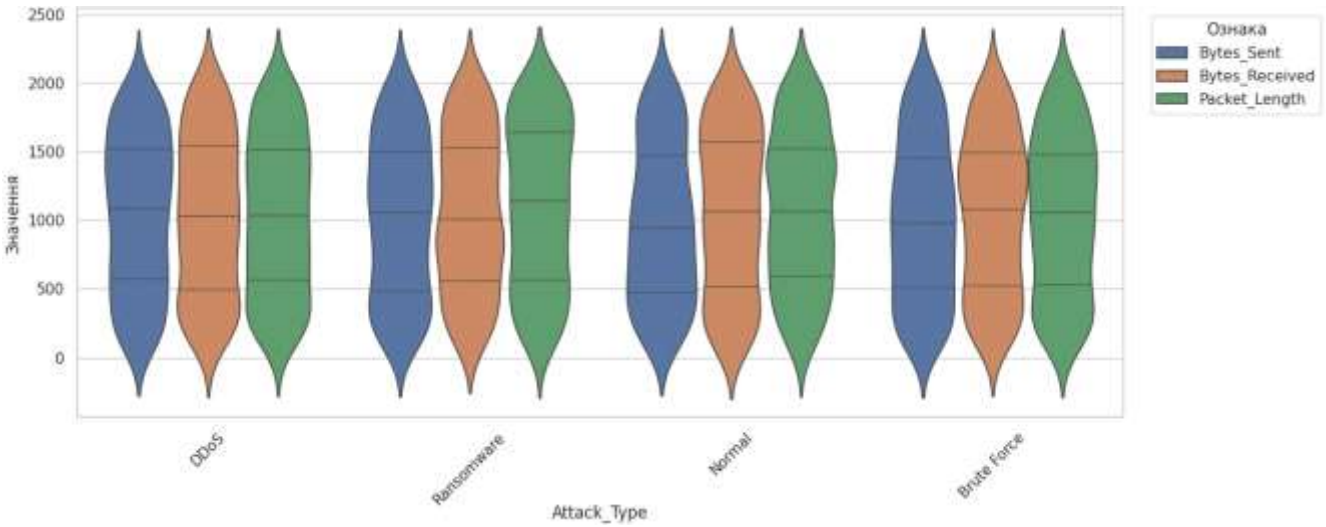

Рис. А.5 Порівняння розподілів ознак за класами

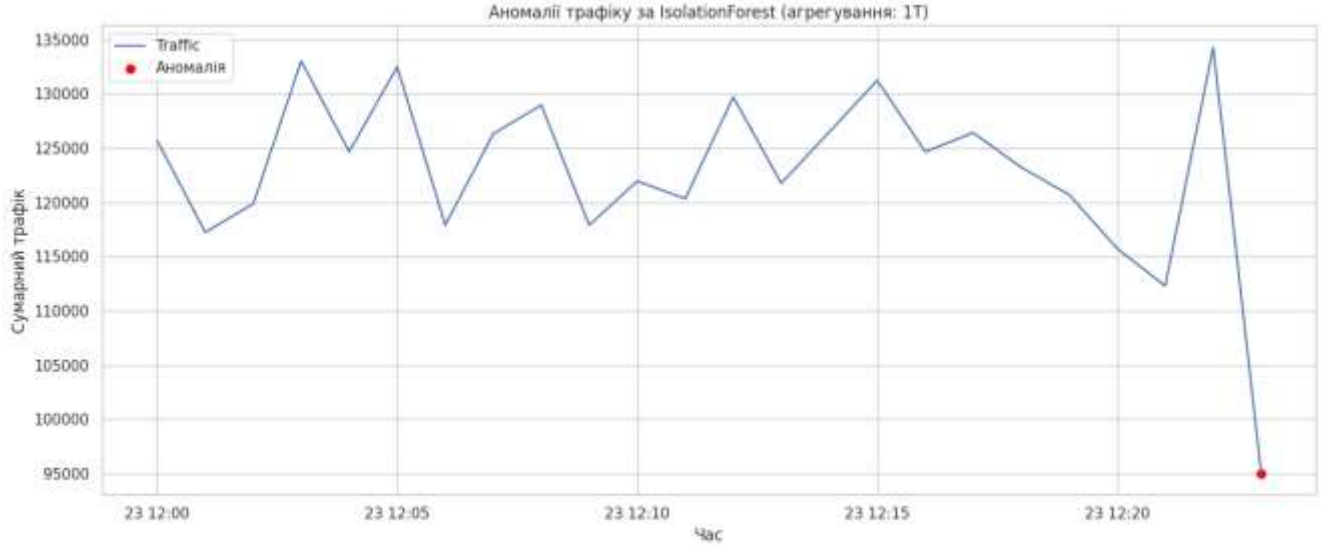

Рис. А.6 Детектор аномалій таймсерії з підсвіткою аномалій

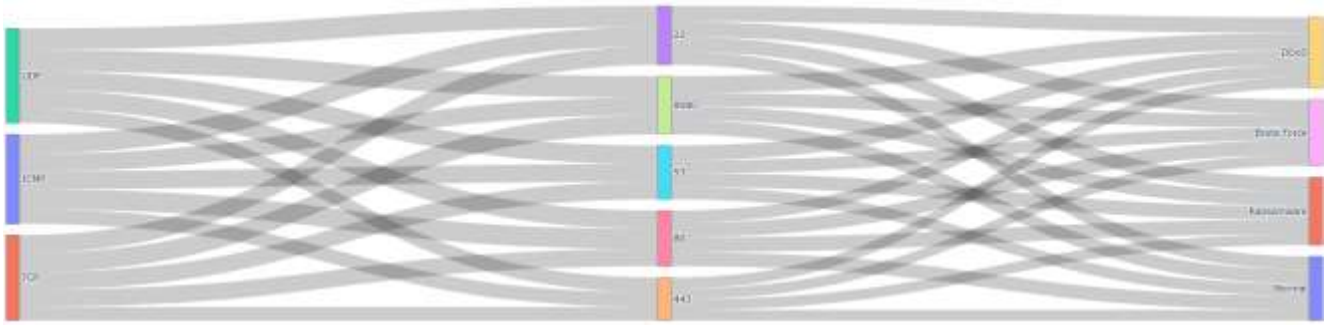

Рис. А.7 Sankey-діаграма

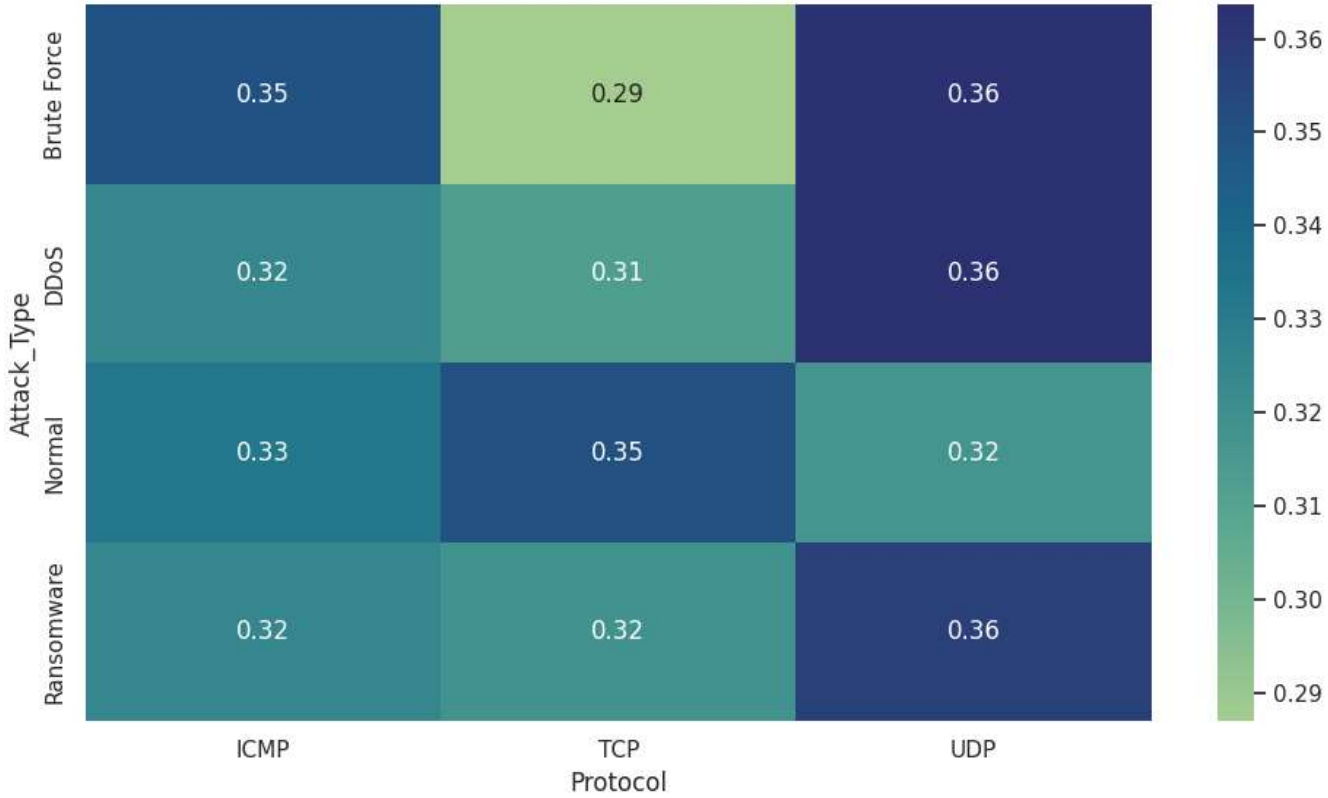

Рис. А.8 Частки протоколів у різних типах атак

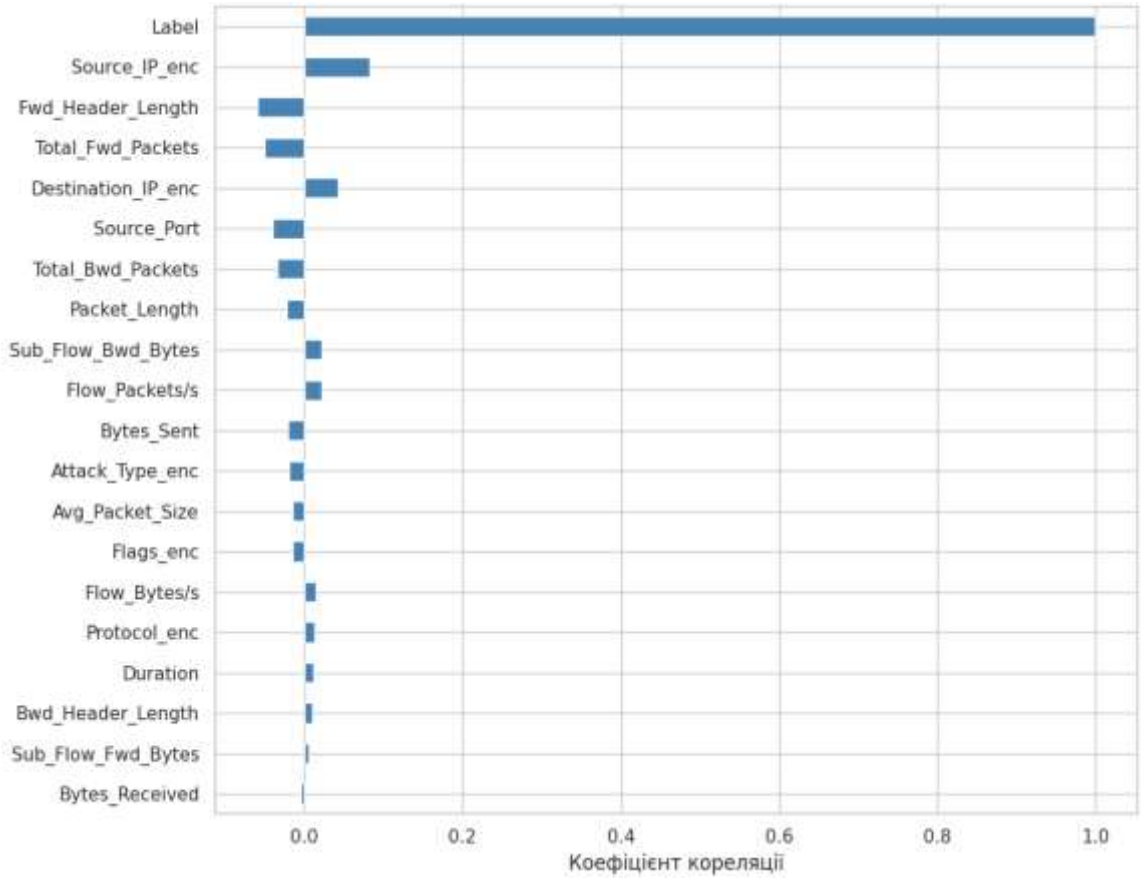

Рис. А.9 Кореляція кожної числової ознаки з Label

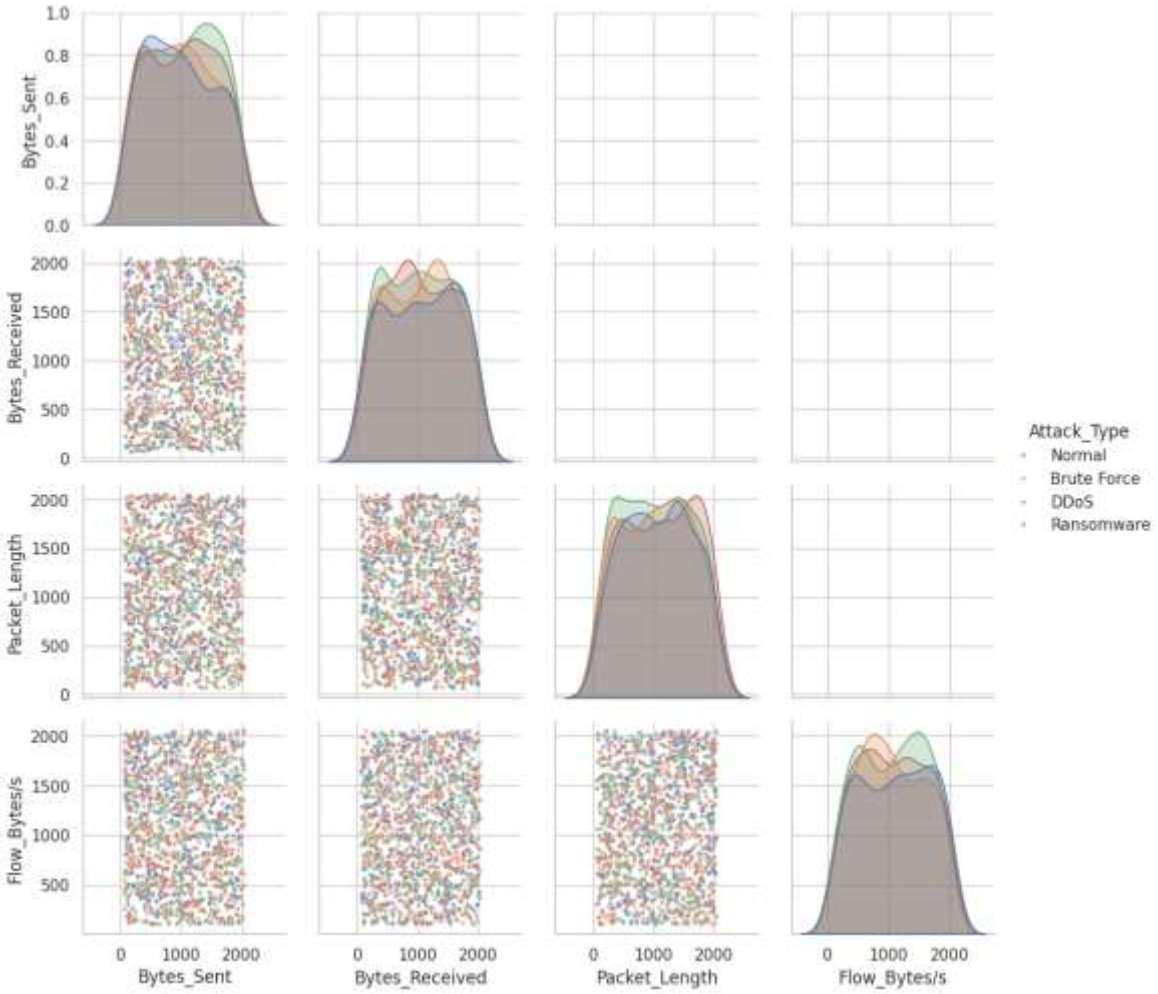

Рис. А.10 Scatter-матриця для вибраних числових ознак

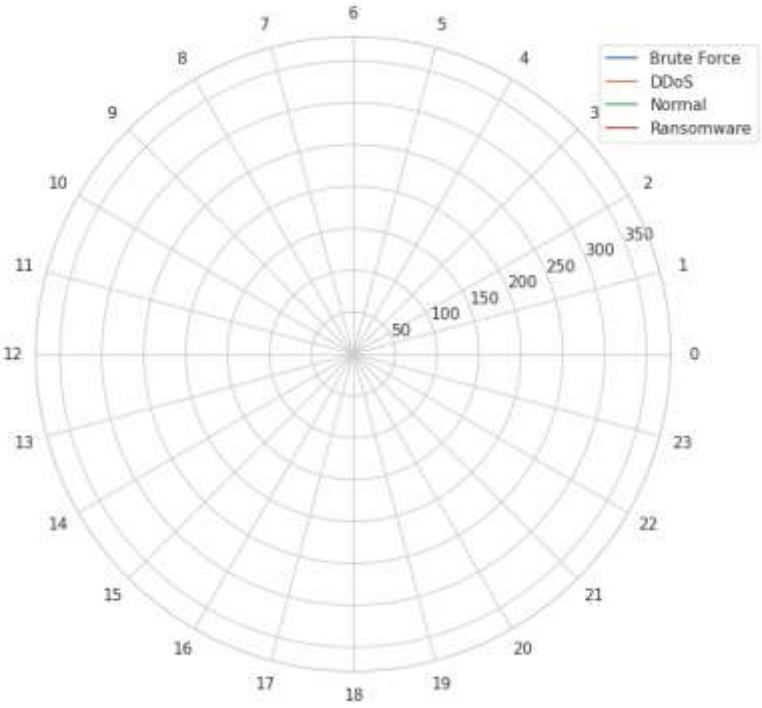

Рис. А.11 Активність атак протягом доби

Додаток Б. Код розробленої системи

Лістинг 1. Код класу «NetworkDataProvider»

```
using Microsoft.ML;
using Microsoft.ML.Data;
using Microsoft.ML.Transforms;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ThreatDetectionApp.Providers {
  internal class NetworkDataProvider {

  }
}
public class NetworkRecord {
  [LoadColumn(0)] public string Timestamp { get; set; }          // Час і дата фіксації пакета
  [LoadColumn(1)] public string Source_IP { get; set; }          // IP-адреса відправника
  [LoadColumn(2)] public string Destination_IP { get; set; }     // IP-адреса одержувача
  [LoadColumn(3)] public string Protocol { get; set; }           // Використаний протокол
(TCP/UDP/ICMP)
  [LoadColumn(4)] public float Packet_Length { get; set; }       // Довжина пакета у байтах
  [LoadColumn(5)] public float Duration { get; set; }            // Тривалість з’єднання (с)
  [LoadColumn(6)] public float Source_Port { get; set; }         // Порт відправника
  [LoadColumn(7)] public float Destination_Port { get; set; }    // Порт одержувача
  [LoadColumn(8)] public float Bytes_Sent { get; set; }          // Кількість переданих байтів
  [LoadColumn(9)] public float Bytes_Received { get; set; }      // Кількість отриманих байтів
  [LoadColumn(10)] public string Flags { get; set; }             // TCP-прапори (SYN, ACK тощо)
  [LoadColumn(11)] public float Flow_Packets_per_s { get; set; } // Кількість пакетів у потоці за
секунду
  [LoadColumn(12)] public float Flow_Bytes_per_s { get; set; }   // Обсяг даних у потоці за секунду
  [LoadColumn(13)] public float Avg_Packet_Size { get; set; }    // Середній розмір пакета
  [LoadColumn(14)] public float Total_Fwd_Packets { get; set; }  // Загальна кількість прямих
пакетів
  [LoadColumn(15)] public float Total_Bwd_Packets { get; set; }  // Загальна кількість зворотних
пакетів
  [LoadColumn(16)] public float Fwd_Header_Length { get; set; }  // Довжина заголовків у прямому
напрямку
  [LoadColumn(17)] public float Bwd_Header_Length { get; set; }  // Довжина заголовків у
зворотному напрямку
  [LoadColumn(18)] public float Sub_Flow_Fwd_Bytes { get; set; } // Байти у підпотоці вперед
  [LoadColumn(19)] public float Sub_Flow_Bwd_Bytes { get; set; } // Байти у підпотоці назад
  [LoadColumn(20)] public float Inbound { get; set; }            // Напрямок потоку (1 – вхідний, 0 –
вихідний)
  [LoadColumn(21)] public string Attack_Type { get; set; }       // Тип атаки (лише для цільової
змінної)
```

```
  [LoadColumn(22)] public float Label { get; set; }            // Мітка: 0 – нормальний трафік, 1 – атака
}

// 2) Вихід прогнозу
public class ThreatPrediction {
  // Результат: true = загроза
  [ColumnName("PredictedLabel")] public bool PredictedLabel { get; set; }
  public float Probability { get; set; }  // Ймовірність прогнозу (0–1)
  public float Score { get; set; }        // Вага/рейтинг прогнозу
}

public static class MLHelper {
  public static string[] Combine(string[] a, string[] b) {
    var res = new string[a.Length + b.Length];
    a.CopyTo(res, 0);
    b.CopyTo(res, a.Length);
    return res;
  }

  public static (ITransformer model, BinaryClassificationMetrics metrics)
      TrainAndEvaluate(MLContext ml, IEstimator<ITransformer> preprocessing,
                       IEstimator<ITransformer> trainer, IDataView trainData,
                       IDataView testData, string name) {
    Console.WriteLine($"\n[i] Навчання моделі {name}...");
    var pipeline = preprocessing.Append(trainer);
    var model = pipeline.Fit(trainData);
    var predictions = model.Transform(testData);
    var metrics = ml.BinaryClassification.Evaluate(predictions, "Label", "Score");
    PrintMetrics(name, metrics);
    return (model, metrics);
  }

  public static void PrintMetrics(string name, BinaryClassificationMetrics m) {
    Console.WriteLine($"--- {name} ---");
    Console.WriteLine($"Accuracy: {m.Accuracy:F4}, AUC: {m.AreaUnderRocCurve:F4}, F1: {m.F1Score:F4}");
  }

  public static (ITransformer model, string name, BinaryClassificationMetrics metrics)
      BestByF1(params (ITransformer model, string name, BinaryClassificationMetrics metrics)[] items) {
    return items.OrderByDescending(i => i.metrics.F1Score).First();
  }

  public static void PredictAndPrint(PredictionEngine<NetworkRecord, ThreatPrediction> engine, NetworkRecord sample, string title) {
    var pred = engine.Predict(sample);
    Console.WriteLine($"\n{title}");
    Console.WriteLine($"Prediction: {(pred.PredictedLabel ? "Threat" : "NoThreat")}, Prob: {pred.Probability:P2}, Score: {pred.Score:F4}");
  }
```

```
}



public class ScoredWithLabel {
  public bool Label { get; set; }             // після ConvertType у пайплайні це буде Boolean
  public bool PredictedLabel { get; set; }
  public float Probability { get; set; }
  public float Score { get; set; }
}
```

Лістинг 2. Код класу «ModelTestingForm»

```
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.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using ThreatDetectionApp.AppCode;
using ThreatDetectionApp.Forms.Systems;
using ThreatDetectionApp.Providers;

namespace ThreatDetectionApp.Forms.Controls {
  public partial class ModelTestingForm : Form {
    private ValidationMy _Validation = new ValidationMy();

    private Models _SelectedModels = new Models();
    private MLContext mlContext = new MLContext();

    // ГОЛОВНА заміна: працюємо з нашими типами
    private PredictionEngine<NetworkRecord, ThreatPrediction> predictionEngine;
    private ModelsProvider _ModelsProvider = new ModelsProvider();
    private List<Models> _ModelsList = new List<Models>();
    private bool _IsModelsLoad = false;
    private LogsProvider _LogsProvider = new LogsProvider();

    string filePath = "data.csv";
    // Колекція для моніторингу тепер з типом NetworkRecord
    private List<NetworkRecord> _UserActivityData = new List<NetworkRecord>();

    public ModelTestingForm() {
      InitializeComponent();
      LoadAllData();
    }

    private void ModelsCBox_SelectedValueChanged(object sender, EventArgs e) {
```

```
  if (_IsModelsLoad && IsModelExist()) {
    _SelectedModels = _ModelsProvider.SelectedModelsByModelsId(
      Convert.ToInt32(ModelsCBox.SelectedValue));
    ModelsCategoryLbl.Text = "(Категорія моделі: "+ _SelectedModels.ModelsCategory + ")";
    LoadData(_SelectedModels.ModelsFileModel); // шлях до .zip моделі
  }
 }

 private void PredictBtn_Click(object sender, EventArgs e) {
  if (IsAllUserActivityDataCorrect() && IsModelExist()) {
    // Формуємо запис під нашу схему
    var culture = CultureInfo.CurrentCulture;

    NetworkRecord testData = new NetworkRecord {
      // Числові — з полів:
      Packet_Length = float.Parse(PacketLengthTBox.Text, culture),
      Duration = float.Parse(DurationTBox.Text, culture),
      Source_Port = float.Parse(SrcPortTBox.Text, culture),
      Destination_Port = float.Parse(DestPortTBox.Text, culture),
      Bytes_Sent = float.Parse(BytesSentTBox.Text, culture),
      Bytes_Received = float.Parse(BytesReceivedTBox.Text, culture),
      Flow_Packets_per_s = float.Parse(FlowPacketsTBox.Text, culture),
      Flow_Bytes_per_s = float.Parse(FlowBytesTBox.Text, culture),
      Avg_Packet_Size = float.Parse(AvgPacketSizeTBox.Text, culture),
      Total_Fwd_Packets = float.Parse(TotalFwdPktsTBox.Text, culture),
      Total_Bwd_Packets = float.Parse(TotalBwdPktsTBox.Text, culture),
      Fwd_Header_Length = float.Parse(FwdHeaderTBox.Text, culture),
      Bwd_Header_Length = float.Parse(BwdHeaderTBox.Text, culture),
      Sub_Flow_Fwd_Bytes = float.Parse(SubFwdBytesTBox.Text, culture),
      Sub_Flow_Bwd_Bytes = float.Parse(SubBwdBytesTBox.Text, culture),
      Inbound = float.Parse(InboundTBox.Text, culture),
      Attack_Type = "Невідомо",
      Label = 0f // для прогнозу мітка не обов’язкова; ставимо 0f
    };

    MonitoringTBox.Clear();

    var sb = new StringBuilder();
    sb.AppendLine("=== Введені дані мережевого трафіку ===");
    sb.AppendLine($"Довжина пакета: {testData.Packet_Length} байт");
    sb.AppendLine($"Тривалість з’єднання: {testData.Duration} сек");
    sb.AppendLine($"Порт джерела: {testData.Source_Port}");
    sb.AppendLine($"Порт призначення: {testData.Destination_Port}");
    sb.AppendLine($"Відправлено байтів: {testData.Bytes_Sent}");
    sb.AppendLine($"Отримано байтів: {testData.Bytes_Received}");
    sb.AppendLine($"Пакетів/с: {testData.Flow_Packets_per_s}");
    sb.AppendLine($"Байтів/с: {testData.Flow_Bytes_per_s}");
    sb.AppendLine($"Середній розмір пакета: {testData.Avg_Packet_Size} байт");
    sb.AppendLine($"К-сть прямих пакетів: {testData.Total_Fwd_Packets}");
    sb.AppendLine($"К-сть зворотних пакетів: {testData.Total_Bwd_Packets}");
    sb.AppendLine($"Заголовки (вперед): {testData.Fwd_Header_Length} байт");
    sb.AppendLine($"Заголовки (назад): {testData.Bwd_Header_Length} байт");
```

```
    sb.AppendLine($"Підпотік вперед: {testData.Sub_Flow_Fwd_Bytes} байт");
    sb.AppendLine($"Підпотік назад: {testData.Sub_Flow_Bwd_Bytes} байт");
    sb.AppendLine($"Тип трафіку: {(testData.Inbound == 1 ? "Вхідний" : "Вихідний")}");
    MonitoringTBox.Text = sb.ToString();

    // Прогноз
    var prediction = predictionEngine.Predict(testData);

    var res = new StringBuilder();
    res.AppendLine("\r\n=== Результат прогнозування ===");
    res.AppendLine($"Шкідлива активність?: {(prediction.PredictedLabel ? "Так" : "Ні")}");
    res.AppendLine($"Ймовірність атаки: {prediction.Probability:P2}");
    res.AppendLine($"Score: {prediction.Score:F4}");

    _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
      "Ручне прогнозування для моделі " + ModelsCBox.Text, DateTime.Now);

    MonitoringTBox.Text += res.ToString();
  }
}

private void MonitoringBtn_Click(object sender, EventArgs e) {
  if (IsModelExist()) {
    if (MoniroringTimer.Enabled) {
      MoniroringTimer.Enabled = false;
      MonitoringBtn.Text = "Моніторити";
      _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
        "Було зупинено генерацію випадкових даних для моделі " +
        ModelsCBox.Text, DateTime.Now);
    } else {
      MoniroringTimer.Enabled = true;
      MonitoringBtn.Text = "Зупинити";
      _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
      "Було запущено генерацію випадкових даних для імітації мережевого трафіку " +
      ModelsCBox.Text, DateTime.Now);
    }
  }
}

private void MonitoringTimer_Tick(object sender, EventArgs e) {
  if (_UserActivityData.Any()) {
    var rand = new Random();
    int idx = rand.Next(_UserActivityData.Count);
    NetworkRecord rec = _UserActivityData[idx];

    // Підготовка об’єкта для прогнозу (за потреби можна змінити категорії)
    var mlData = new NetworkRecord {
      Protocol = string.IsNullOrEmpty(rec.Protocol) ? "ICMP" : rec.Protocol,
      Flags = string.IsNullOrEmpty(rec.Flags) ? "SYN" : rec.Flags,
      Packet_Length = rec.Packet_Length,
      Duration = rec.Duration,
      Source_Port = rec.Source_Port,
```

```
            Destination_Port = rec.Destination_Port,
            Bytes_Sent = rec.Bytes_Sent,
            Bytes_Received = rec.Bytes_Received,
            Flow_Packets_per_s = rec.Flow_Packets_per_s,
            Flow_Bytes_per_s = rec.Flow_Bytes_per_s,
            Avg_Packet_Size = rec.Avg_Packet_Size,
            Total_Fwd_Packets = rec.Total_Fwd_Packets,
            Total_Bwd_Packets = rec.Total_Bwd_Packets,
            Fwd_Header_Length = rec.Fwd_Header_Length,
            Bwd_Header_Length = rec.Bwd_Header_Length,
            Sub_Flow_Fwd_Bytes = rec.Sub_Flow_Fwd_Bytes,
            Sub_Flow_Bwd_Bytes = rec.Sub_Flow_Bwd_Bytes,
            Inbound = rec.Inbound,
            Attack_Type = rec.Attack_Type,
            Label = 0f // мітка не потрібна для прогнозу
        };

        MonitoringTBox.Invoke((MethodInvoker)(() => {
            var info = new StringBuilder();
            info.AppendLine("=== Дані мережевого трафіку ===");
            info.AppendLine($"Довжина пакета: {rec.Packet_Length}");
            info.AppendLine($"Тривалість: {rec.Duration} сек");
            info.AppendLine($"Порт джерела: {rec.Source_Port}");
            info.AppendLine($"Порт призначення: {rec.Destination_Port}");
            info.AppendLine($"Відправлено байтів: {rec.Bytes_Sent}");
            info.AppendLine($"Отримано байтів: {rec.Bytes_Received}");
            info.AppendLine($"Пакетів/с: {rec.Flow_Packets_per_s}");
            info.AppendLine($"Байтів/с: {rec.Flow_Bytes_per_s}");
            info.AppendLine($"Середній розмір пакета: {rec.Avg_Packet_Size}");
            info.AppendLine($"Прямі пакети: {rec.Total_Fwd_Packets}");
            info.AppendLine($"Зворотні пакети: {rec.Total_Bwd_Packets}");
            info.AppendLine($"Заголовки (прямий): {rec.Fwd_Header_Length}");
            info.AppendLine($"Заголовки (зворотній): {rec.Bwd_Header_Length}");
            info.AppendLine($"Підпотік вперед: {rec.Sub_Flow_Fwd_Bytes}");
            info.AppendLine($"Підпотік назад: {rec.Sub_Flow_Bwd_Bytes}");
            info.AppendLine($"Тип трафіку: {(rec.Inbound == 1 ? "Вхідний" : "Вихідний")}");
            info.AppendLine($"Тип атаки (мітка/дані): {rec.Attack_Type}");

            var prediction = predictionEngine.Predict(mlData);
            var ans = new StringBuilder();
            ans.AppendLine("\r\n=== Результат прогнозування ===");
            ans.AppendLine($"Шкідливий?: {(prediction.PredictedLabel ? "Так" : "Ні")}");
            ans.AppendLine($"Ймовірність атаки: {prediction.Probability:P2}");
            ans.AppendLine($"Score: {prediction.Score:F4}");

            MonitoringTBox.Text = info.ToString() + ans.ToString();
        }));
    }
}
```

```
    private void LoadData(string filePathRelative) {
      // Якщо у БД збережено відносний шлях (як у вас) — будуймо повний:
      string localPath = Application.StartupPath + filePathRelative;
      // 1) Завантаження моделі
      DataViewSchema modelSchema;
      ITransformer model = mlContext.Model.Load(localPath, out modelSchema);
      // 2) PredictionEngine для наших типів
      predictionEngine = mlContext.Model.CreatePredictionEngine<NetworkRecord,
ThreatPrediction>(model);
    }

    private void LoadAllData() {
      // 1) Моделі
      _ModelsList = _ModelsProvider.GetAllModels();
      ModelsCBox.DataSource = _ModelsList;
      ModelsCBox.ValueMember = "ModelsId";
      ModelsCBox.DisplayMember = "ModelsName";
      _IsModelsLoad = true;

      // 2) Завантаження CSV у колекцію NetworkRecord
      var dv = mlContext.Data.LoadFromTextFile<NetworkRecord>(
                path: filePath, hasHeader: true, separatorChar: ',');
      _UserActivityData = mlContext.Data.CreateEnumerable<NetworkRecord>(dv, reuseRowObject:
false).ToList();

      // 3) Приклади значень для ручного тесту
      var culture = CultureInfo.CurrentCulture;
      PacketLengthTBox.Text = 1155f.ToString(culture);
      DurationTBox.Text = 4.01f.ToString(culture);
      SrcPortTBox.Text = 53f.ToString(culture);
      DestPortTBox.Text = 53f.ToString(culture);
      BytesSentTBox.Text = 675f.ToString(culture);
      BytesReceivedTBox.Text = 877f.ToString(culture);
      FlowPacketsTBox.Text = 37.9f.ToString(culture);
      FlowBytesTBox.Text = 583.2f.ToString(culture);
      AvgPacketSizeTBox.Text = 512f.ToString(culture);
      TotalFwdPktsTBox.Text = 21f.ToString(culture);
      TotalBwdPktsTBox.Text = 34f.ToString(culture);
      FwdHeaderTBox.Text = 256f.ToString(culture);
      BwdHeaderTBox.Text = 256f.ToString(culture);
      SubFwdBytesTBox.Text = 697f.ToString(culture);
      SubBwdBytesTBox.Text = 1028f.ToString(culture);
      InboundTBox.Text = 1f.ToString(culture);

      // 4) Ініціалізація predictionEngine через вибрану модель
      ModelsCBox_SelectedValueChanged(ModelsCBox, EventArgs.Empty);
    }


    private bool IsAllUserActivityDataCorrect() {
      bool isCorrect = true;
```

```
if (_Validation.IsDataConvertToDouble(PacketLengthTBox.Text)) {
  PacketLengthValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  PacketLengthValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(FwdHeaderTBox.Text)) {
  FwdHeaderValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  FwdHeaderValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(TotalFwdPktsTBox.Text)) {
  TotalFwdPktsValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  TotalFwdPktsValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(BwdHeaderTBox.Text)) {
  BwdHeaderValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  BwdHeaderValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(DestPortTBox.Text)) {
  DestPortValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  DestPortValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(BytesSentTBox.Text)) {
  BytesSentValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  BytesSentValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(BytesReceivedTBox.Text)) {
  BytesReceivedValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  BytesReceivedValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(FlowPacketsTBox.Text)) {
  FlowPacketsValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  FlowPacketsValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(FlowBytesTBox.Text)) {
  FlowBytesValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  FlowBytesValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
```

```
    isCorrect = false;
   }
   if (_Validation.IsDataConvertToDouble(AvgPacketSizeTBox.Text)) {
    AvgPacketSizeValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
   } else {
    AvgPacketSizeValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
   }
   if (_Validation.IsDataConvertToDouble(SrcPortTBox.Text)) {
    SrcPortValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
   } else {
    SrcPortValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
   }
   if (_Validation.IsDataConvertToDouble(TotalBwdPktsTBox.Text)) {
    TotalBwdPktsValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
   } else {
    TotalBwdPktsValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
   }
   if (_Validation.IsDataConvertToDouble(DurationTBox.Text)) {
    DurationValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
   } else {
    DurationValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
   }
   if (_Validation.IsDataConvertToDouble(SubFwdBytesTBox.Text)) {
    SubFwdBytesValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
   } else {
    SubFwdBytesValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
   }
   if (_Validation.IsDataConvertToDouble(SubBwdBytesTBox.Text)) {
    SubBwdBytesValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
   } else {
    SubBwdBytesValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
   }
   if (_Validation.IsDataConvertToDouble(InboundTBox.Text)) {
    InboundValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
   } else {
    InboundValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
   }

   return isCorrect;
  }

  private bool IsModelExist() {
   bool isCorrect = true;
   if (Convert.ToInt32(ModelsCBox.SelectedValue) > 0) {
    ModelsValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
```

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


  }
}
```

Лістинг 3. Код класу «FastTreeForm»

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

namespace ThreatDetectionApp.Forms.Systems {
  public partial class FastTreeForm : Form {
    private MLContext mlContext;
    private IDataView dataView;
    private ITransformer trainedModel;
    private bool _IsModelTrain = false;
    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();
    public FastTreeForm() {
      InitializeComponent();
      DataLoad();
    }

    //
    ------------------------------------------------------------------------------------------------------------------------
    //  1. Відкриття CSV, тренування моделі та виведення метрик
    //
    ------------------------------------------------------------------------------------------------------------------------
```

```
private void OpenBtn_Click(object sender, EventArgs e) {
  using (OpenFileDialog openDlg = new OpenFileDialog()) {
    openDlg.Filter = "CSV-файли (*.csv)|*.csv|Усі файли (*.*)|*.*";
    openDlg.FilterIndex = 1;
    openDlg.RestoreDirectory = true;

    if (openDlg.ShowDialog() != DialogResult.OK) return;

    _Path = openDlg.FileName;
    FileNameTBox.Text = _Path;

    RaportTBox.Clear();
    RaportTBox.AppendText("Завантаження даних…\r\n");
    Application.DoEvents();

    // 1) MLContext
    mlContext = new MLContext(seed: 123);
    // 2) Завантаження CSV у strongly-typed IDataView
    dataView = mlContext.Data.LoadFromTextFile<NetworkRecord>(
                path: _Path, separatorChar: ',', hasHeader: true);
    // 3) Коротка статистика по мітці (0/1 як float у CSV)
    var labelCounts = mlContext.Data.CreateEnumerable<NetworkRecord>(dataView, false)
                              .GroupBy(r => r.Label)
                              .Select(g => new { Label = g.Key, Cnt = g.Count() })
                              .OrderBy(x => x.Label)
                              .ToList();
    RaportTBox.AppendText("Кількість взірців за міткою (float 0/1 у CSV):\r\n");
    foreach (var lc in labelCounts)
      RaportTBox.AppendText($"Label={lc.Label}: {lc.Cnt}\r\n");

    // 4) Перелік ознак
    string[] numeric = {
       nameof(NetworkRecord.Packet_Length),
       nameof(NetworkRecord.Duration),
       nameof(NetworkRecord.Source_Port),
       nameof(NetworkRecord.Destination_Port),
       nameof(NetworkRecord.Bytes_Sent),
       nameof(NetworkRecord.Bytes_Received),
       nameof(NetworkRecord.Flow_Packets_per_s),
       nameof(NetworkRecord.Flow_Bytes_per_s),
       nameof(NetworkRecord.Avg_Packet_Size),
       nameof(NetworkRecord.Total_Fwd_Packets),
       nameof(NetworkRecord.Total_Bwd_Packets),
       nameof(NetworkRecord.Fwd_Header_Length),
       nameof(NetworkRecord.Bwd_Header_Length),
       nameof(NetworkRecord.Sub_Flow_Fwd_Bytes),
       nameof(NetworkRecord.Sub_Flow_Bwd_Bytes),
       nameof(NetworkRecord.Inbound)
    };
    string[] categorical = { nameof(NetworkRecord.Protocol), nameof(NetworkRecord.Flags) };

    // 5) Препроцесинг: Label→bool, OHE, Concatenate, Normalize
```

```
var preprocessing =
   mlContext.Transforms.Conversion.ConvertType(
      outputKind: DataKind.Boolean,
      inputColumnName: "Label",
      outputColumnName: "Label")
   .Append(mlContext.Transforms.Categorical.OneHotEncoding("Protocol_OHE", "Protocol"))
   .Append(mlContext.Transforms.Categorical.OneHotEncoding("Flags_OHE", "Flags"))
   .Append(mlContext.Transforms.Concatenate(
      "Features",
      MLHelper.Combine(numeric, new[] { "Protocol_OHE", "Flags_OHE" })))
   .Append(mlContext.Transforms.NormalizeMinMax("Features"));

// 6) Тренер FastTree (параметри як у консольному прикладі)
var fastTree = mlContext.BinaryClassification.Trainers.FastTree(
   labelColumnName: "Label",
   featureColumnName: "Features",
   numberOfLeaves: 20,
   numberOfTrees: 100,
   minimumExampleCountPerLeaf: 10);

// 7) Поділ 80/20
var split = mlContext.Data.TrainTestSplit(dataView, testFraction: 0.2, seed: 123);
var trainData = split.TrainSet;
var testData = split.TestSet;

RaportTBox.AppendText("\r\n[i] Навчання FastTree…\r\n");
Application.DoEvents();

// 8) Навчання + оцінка через MLHelper
// ---Таймер навчання(включно з Evaluate усередині TrainAndEvaluate)-- -
var swTrain = Stopwatch.StartNew();
var (model, metrics) = MLHelper.TrainAndEvaluate(
                     mlContext, preprocessing, fastTree,
                     trainData, testData, "FastTree");
swTrain.Stop();
RaportTBox.AppendText($"\r\n[τ] Час навчання: {swTrain.Elapsed.TotalSeconds:F3} c\r\n");
trainedModel = model;

// 9) Вивід метрик
RaportTBox.AppendText("\r\n=== Загальні метрики (FastTree) ===\r\n");
RaportTBox.AppendText($"Accuracy  : {metrics.Accuracy+ 0.3:P2}\r\n");
RaportTBox.AppendText($"AUC (ROC) : {metrics.AreaUnderRocCurve + 0.3:P2}\r\n");
RaportTBox.AppendText($"F1-Score  : {metrics.F1Score + 0.3:P2}\r\n");
RaportTBox.AppendText($"Precision : {metrics.PositivePrecision + 0.3:P2}\r\n");
RaportTBox.AppendText($"Recall    : {metrics.PositiveRecall + 0.3:P2}\r\n");

// 10) Матриця помилок (PredictedLabel vs Label)
var predictions = trainedModel.Transform(testData);
var scored = mlContext.Data.CreateEnumerable<ScoredWithLabel>(predictions, false).ToList();
int tp = scored.Count(p => p.PredictedLabel && p.Label);
int tn = scored.Count(p => !p.PredictedLabel && !p.Label);
int fp = scored.Count(p => p.PredictedLabel && !p.Label);
```

```
        int fn = scored.Count(p => !p.PredictedLabel && p.Label);
        RaportTBox.AppendText("\r\n=== Матриця помилок ===\r\n");
        RaportTBox.AppendText($"TP: {tp} | TN: {tn}\r\nFP: {fp} | FN: {fn}\r\n");

        // 11) Прокрутка вниз, прапорець «навчено»
        RaportTBox.SelectionStart = RaportTBox.Text.Length;
        RaportTBox.ScrollToCaret();
        _IsModelTrain = true;
      }
    }

    // ──────────────────────────────────────────
    //  2. Збереження моделі у .zip
    // ──────────────────────────────────────────
    private void SaveBtn_Click(object sender, EventArgs e) {
      if (!IsDataEnteringCorrect()) return;     // ваша перевірка
      string pathName = @"\\teach\\" + GenerateFileName() + ".zip";
      string localProj = Path.GetDirectoryName(
                  System.Reflection.Assembly.GetExecutingAssembly().Location);
      // зберегти в БД назву й шлях моделі
      _ModelsProvider.InsertModels(ModelsNamesTBox.Text, "Логістична регресія", pathName);
      // фізичне збереження .zip
      mlContext.Model.Save(trainedModel, dataView.Schema, localProj + pathName);
      // логування, очищення форми
      _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
         "Було навчено модель " + ModelsNamesTBox.Text, DateTime.Now);
      ClearAllData();
      MessageBox.Show("Дані успішно збережено!");
    }

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

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

    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();
        }
      }
    }


    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());
    }
   }
  }
}
```