НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп'ютерних систем, мереж та кібербезпеки

_____ Касаткін Д.Ю., к.пед.н., доц.

(підпис) (ПІБ, вчене звання і ступінь)

«___»_____2025 р.

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

На тему: «Забезпечення безпеки при інтеграції нових технологій у існуючі комп'ютерні системи університету»

Спеціальність 123 «Комп'ютерна інженерія»

_____

Гарант освітньої програми

к.ф. м н., доцент _____ / Нікітенко Є.В. /

(підпис) (ПІБ)

Керівник дипломного проекту: _____ / Лахно В.А. /

(підпис) (ПІБ)

Виконав: _____ / Гайдук М.В./

(підпис) (ПІБ)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«ЗАТВЕРДЖУЮ»
завідувач кафедри
комп'ютерних систем, мереж та кібербезпеки
_____/ Касаткін Д.Ю., к.пед.н., доц. /
(підпис) (ПІБ, вчене звання і ступінь)
«__» _____ 20__ р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ БАКАЛАВРСЬКОЇ СТУДЕНТУ

Гайдук Максим Валерійович
(прізвище, ім'я, по батькові)
Спеціальність (напрям підготовки): комп'ютерна інженерія
Тема кваліфікаційної бакалаврської роботи: «
затверджена наказом ректора НУБіП України від " "_____202р. №___« »___
Термін подання завершеної роботи на кафедру
Вихідні дані до кваліфікаційної бакалаврської роботи Забезпечення безпеки при інтеграції нових технологій у існуючі комп'ютерні системи університету

Перелік питань, що підлягають розробці:
1. Аналіз предметної області
2. Проєктування програмного забезпечення
3. Реалізація програмного забезпечення
4. Тестування комп'ютерної системи

Перелік графічного матеріалу (за потреби)

Дата видачі завдання "_____"_____2025 р.

Керівник кваліфікаційної роботи _____ ___Лахно В.А., д.т.н.
(підпис) (прізвище та ініціали)
Завдання прийняв до виконання _____ _____Гайдук М.В._____
(підпис) (прізвище та ініціали студента)

РЕФЕРАТ

Пояснювальна записка: 89 сторінок, 40 рисунків, 6 таблиць, 1 додаток, 21 джерело.

МАШИННЕ НАВЧАННЯ, МЕРЕЖЕВА БЕЗПЕКА, АНАЛІЗ ТРАФІКУ, C#, ML.NET.

Об'єкт дослідження – процес виявлення аномалій у мережевому трафіку комп’ютерної системи університету в умовах інтеграції нових інформаційних технологій.

Метою роботи є розробка програмного забезпечення для виявлення аномального трафіку з використанням методів машинного навчання, аналіз сучасних загроз безпеці в освітньому середовищі, а також оцінка точності і продуктивності запропонованої системи.

Проєкт складається з чотирьох розділів.

У першому розділі проаналізовано сучасні загрози безпеці, методи безінтрузивного аналізу та алгоритми машинного навчання для виявлення аномалій. Обґрунтовано вибір мови C# і середовища Visual Studio 2022.

Другий розділ містить проєктну частину: побудовано блок-схеми, діаграми класів трьохрівневої архітектури та сформовано математичну модель на базі FastTree.

У третьому розділі реалізовано модулі обробки трафіку, машинного навчання та захисту даних, інтегровано ML.NET і створено інтерфейс користувача.

Четвертий розділ присвячено тестуванню: отримано точність 94,21 %, AUC 99,16 %, F1 Score 93,24 %; проаналізовано продуктивність і стабільність роботи системи.

У результаті виконання дипломної роботи розроблено, протестовано й оцінено програмне забезпечення виявлення аномалій у мережевому трафіку на основі машинного навчання, що може бути використане для посилення кібербезпеки в освітньому середовищі.

ЗМІСТ

ВСТУП

У сучасну епоху стрімкої цифровізації заклади вищої освіти дедалі частіше опиняються перед необхідністю інтеграції новітніх технологій у вже наявну інформаційну інфраструктуру. Таке оновлення обумовлюється не лише прагненням до інновацій, але й вимогами якості освітніх послуг, безперервної комунікації, віддаленого доступу до навчальних матеріалів, а також забезпечення функціонування дослідницьких проєктів на базі інституцій. З одного боку, широке впровадження хмарних сервісів, систем управління навчальним процесом, інтернету речей (IoT) і штучного інтелекту створює нові можливості для автоматизації, персоналізації та розширення академічного середовища [1]. З іншого боку, ці самі технології істотно ускладнюють ландшафт кібербезпеки, генеруючи нові поверхні атак, які не були враховані на етапі проєктування традиційних комп'ютерних систем [2].

Інтеграція нових технологій у комп'ютерну систему без належної уваги до безпеки може призвести до критичних наслідків: витоку персональних даних студентів і викладачів, порушення доступу до навчальних платформ, компрометації систем керування дослідницькою інформацією, а в окремих випадках – до зупинки життєво важливих освітніх процесів. На тлі геополітичних викликів, що постали перед Україною, ці загрози набувають ще більшої актуальності, оскільки сфера освіти дедалі частіше стає об'єктом інформаційного та кібернетичного впливу.

Існуючі засоби захисту, такі як класичні антивірусні рішення, фаєрволи або ізольовані сегменти мережі, виявляються недостатніми у випадку із складними, гібридними архітектурами. У цьому контексті виникає потреба у створенні адаптивних, контекстно-орієнтованих механізмів забезпечення безпеки, які враховують динаміку інтеграційних процесів. Досвід провідних європейських та американських університетів демонструє ефективність підходів, що базуються на інтеграції систем виявлення вторгнень, застосуванні методів машинного навчання для аналізу трафіку та впровадженні принципів

Zero Trust [3]. Однак пряма імплементація таких моделей у вітчизняних умовах часто виявляється неможливою через обмежені ресурси, специфіку локальної інфраструктури та нормативні обмеження, пов’язані із захистом персональних даних.

Таким чином, постає завдання розробки гнучкої методології інтеграції нових технологій з урахуванням вимог до безпеки, яка була б адаптована до умов українських університетів. Це передбачає аналіз архітектурної сумісності, формалізацію типових загроз, оцінку ризиків і впровадження механізмів активного моніторингу стану системи у режимі реального часу. Важливо не лише визначити надійні технологічні рішення, а й забезпечити їхню прозору інтеграцію в існуючі системи без порушення стабільності та з урахуванням обмеженого ІТ-персоналу, характерного для більшості ЗВО.

Наукова й практична цінність дослідження полягає у формуванні цілісного підходу до безпечної інтеграції інноваційних ІТ-рішень у критичну інфраструктуру навчального закладу, що відповідає сучасним викликам кіберпростору. Результати такої роботи можуть бути застосовані не лише в університетах, але й в інших державних установах, які поступово переходять на сучасні цифрові платформи. Отже, комплексне вивчення та впровадження засобів захисту під час інтеграції нових технологій є вкрай необхідним кроком для забезпечення кіберстійкості освітнього сектору України.

Метою роботи є розробка програмного забезпечення для виявлення аномального трафіку з використанням методів машинного навчання, аналіз сучасних загроз безпеці в освітньому середовищі, а також оцінка точності і продуктивності запропонованої системи.

Для реалізації поставленої мети необхідно вирішити наступні задачі:

– проаналізувати характерні загрози, що виникають під час застосування інформаційних технологій в інфраструктурі університетських комп’ютерних систем, та обґрунтувати потребу в засобах моніторингу й виявлення аномалій;

– розробити архітектурну структуру програмного забезпечення, що забезпечує обробку вхідних логів і трафіку, а також інтеграцію алгоритмів машинного навчання для виявлення нетипової активності;

– реалізувати програмний модуль, що включає компоненти збору даних, передобробки, класифікації аномалій і формування повідомлень безпеки з урахуванням особливостей університетської ІТ-інфраструктури;

– здійснити тестування програмного модуля з використанням змодельованих або реальних даних, оцінити його працездатність, функціональну повноту та коректність виявлення відхилень у мережевому трафіку.

Об'єкт дослідження – процес виявлення аномалій у мережевому трафіку комп’ютерної системи університету в умовах інтеграції нових інформаційних технологій.

Предмет дослідження – програмно-алгоритмічні засоби виявлення аномальної мережевої активності як інструмент забезпечення безпеки в умовах модернізації інформаційної інфраструктури університету.

Практична цінність роботи полягає у створенні програмного модуля, що дозволяє виявляти потенційно небезпечну мережеву активність під час інтеграції нових технологій у комп’ютерні системи університету. Розроблене рішення може бути використане для підвищення рівня контролю за інформаційними потоками без потреби у суттєвій зміні існуючої інфраструктури.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Сучасні загрози безпеці при впровадженні нових ІТ-рішень

Інтенсивна цифрова трансформація освітніх установ зумовлює потребу у впровадженні новітніх інформаційних технологій, що стають критично важливими для забезпечення ефективного функціонування університетських систем управління, навчального процесу та наукових досліджень. Проте такі процеси супроводжуються появою нового спектра кіберзагроз, обумовлених не лише технічною складністю впроваджуваних рішень, а й неоднорідністю їх інтеграції з наявною інформаційною інфраструктурою. З огляду на це, особливої ваги набуває класифікація загроз, що виникають унаслідок кібератак та поширення шкідливого програмного забезпечення [4]. На рис. 1.1 подано структуру найбільш поширених векторів атак, які становлять серйозну загрозу для інформаційних систем університетів, передусім у процесі впровадження нових ІТ-рішень.

Рисунок 1.1 – Види кібератак та шкідливого програмного забезпечення

Одним із поширених векторів кібератак є використання месенджерів та електронної пошти як каналів доставки шкідливого програмного забезпечення або фішингових повідомлень. Застосування методів соціальної інженерії

дозволяє зловмисникам створювати повідомлення, які імітують комунікацію з довірених джерел, що спонукає користувачів відкривати заражені вкладення або переходити за посиланнями на шкідливі вебресурси. Прикладом таких дій є цілеспрямовані кампанії розсилання електронних листів, що візуально відтворюють офіційні повідомлення університетських адміністрацій або наукових установ, містячи вкладення, які активують шкідливі скрипти після відкриття.

Серед критичних загроз також виокремлюються атаки, засновані на експлуатації вразливостей нульового дня (zero-day), що існують у програмному забезпеченні до моменту випуску офіційних оновлень [5]. Такі атаки особливо небезпечні через неможливість їхнього виявлення стандартними антивірусними засобами. У 2021 році прикладом подібної загрози стала вразливість у поштовому сервері Microsoft Exchange, яка була використана для отримання несанкціонованого віддаленого доступу до інформаційних систем численних університетів та наукових інститутів [6].

Широкого поширення набули шкідливі програми типу вірусів і троянів, які потрапляють у систему через інсталяцію фальшивих оновлень програмного забезпечення або завантаження файлів із ненадійних джерел. Зокрема, під виглядом оновлень для таких програм як Adobe Reader або Zoom часто маскуються троянські програми, що відкривають віддалений доступ до системи користувача.

Особливу загрозу становить програмне забезпечення класу ransomware, яке здійснює шифрування критичних файлів або блокує доступ до систем із подальшою вимогою викупу. Університетські установи є вразливими до таких атак через зосередження значних обсягів персональних і наукових даних. Одним із відомих випадків стала атака на Maastricht University (Нідерланди), у результаті якої було зашифровано понад 250 серверів, включно з науковими архівами, що спричинило суттєві перебої у функціонуванні закладу [7].

Окрім шкідливого програмного забезпечення, надзвичайно серйозною загрозою для комп’ютерних систем університету залишається

несанкціонований доступ до даних, особливо в умовах активної взаємодії локальних серверів з хмарними сервісами, відкритими API та мобільними клієнтами. Більшість порушень конфіденційності або цілісності в освітньому середовищі пов’язані саме з людськими помилками у налаштуваннях безпеки, неправильним управлінням обліковими даними та використанням небезпечних каналів зв’язку. На рис. 1.2 узагальнено основні сценарії, що призводять до несанкціонованого доступу до даних у сучасних IT-системах.

Рисунок 1.2 – Види несанкціонованого доступу до даних

Серед поширених причин несанкціонованого доступу до систем однією з ключових є використання слабких або типових паролів, що легко піддаються атакам типу перебору (brute-force) або словникових атак (dictionary attacks). Додаткові ризики виникають у разі повторного використання паролів у кількох інформаційних системах, особливо якщо хоча б одна з них зазнає витоку даних. Ілюстрацією таких випадків є злами облікових записів викладачів у системах дистанційного навчання (LMS), де паролі дублювали ідентифікатори від корпоративної електронної пошти або персональних сервісів.

Проблема безпечного зберігання та передавання інформації залишається однією з найгостріших у сфері захисту даних. Паролі, токени доступу або інша конфіденційна інформація іноді передаються у відкритому вигляді між

вузловими серверами та хмарними платформами, а також зберігаються без належного криптографічного захисту. Використання застарілих криптографічних алгоритмів, таких як MD5 чи SHA-1, значно підвищує вразливість систем до атак, зокрема до колізій та підстановок.

До переліку суттєвих загроз належать також атаки типу «людина посередині» (MITM), які виникають у випадках відсутності належного захисту з'єднання (TLS/SSL) або при підключенні користувачів до неавтентичних Wi-Fi-точок доступу [8]. У таких умовах зловмисники можуть здійснювати перехоплення автентифікаційних даних, модифікувати інформацію в процесі передачі або повністю підмінювати сеанси взаємодії з системою. Університетські мережі, зокрема гостьові зони та загальнодоступні підключення, часто стають ціллю подібних атак.

Вразливості можуть виникати також унаслідок неконтрольованого доступу до API-інтерфейсів, коли не реалізовано механізмів IP-фільтрації, автентифікації або авторизації запитів. Така недбалість у налаштуванні безпеки може дозволити стороннім особам отримувати доступ до конфіденційної інформації через відкриті кінцеві точки. Відомий випадок подібної уразливості зафіксовано в одному з електронних щоденників, де через неналежно захищене API можна було отримати доступ до оцінок студентів, використовуючи лише їхній унікальний ідентифікатор [9].

Особливу загрозу для університетських комп'ютерних систем у процесі інтеграції нових технологій становлять IoT-пристрої (Internet of Things), які дедалі частіше використовуються в освітньому середовищі – від систем відеоспостереження та клімат-контролю до сенсорів у лабораторному обладнанні. Попри свою функціональну цінність, такі пристрої часто характеризуються низьким рівнем захисту, спрощеною прошивкою та відсутністю регулярного оновлення, що створює значну кількість нових векторів атак. На рис. 1.3 наведено класифікацію основних загроз безпеці, пов'язаних із використанням IoT-пристроїв в IT-інфраструктурі університету.

Рисунок 1.3 – Загрози IoT-пристроїв

Суттєвим чинником зниження безпеки інтернет-речей залишається використання стандартних або типових облікових даних, які не змінюються після первинного налаштування пристрою (наприклад, login: admin, password: admin). Такий підхід відкриває можливість несанкціонованого доступу до інтерфейсів управління мережевим обладнанням, камерами відеоспостереження, маршрутизаторами чи сенсорами. Значного розголосу набув інцидент із масовим зламом тисяч IP-камер, які були об’єднані в ботнет Mirai через незмінені паролі за замовчуванням [10].

Недостатність механізмів автентифікації та шифрування під час передачі даних між пристроями та серверною інфраструктурою створює передумови для несанкціонованого перехоплення інформації. Уразливими до таких атак залишаються відеопотоки, телеметричні дані (наприклад, температурні показники) або керувальні команди до виконавчих пристроїв. Найбільшої шкоди це завдає в умовах використання відкритих Wi-Fi-мереж, поширених у навчальних закладах.

Поширеною технічною проблемою є відсутність регулярних оновлень мікропрограмного забезпечення або використання застарілих версій прошивок із відомими вразливостями. В IT-підрозділах університетів оновлення часто ігноруються через велику кількість пристроїв, обмежений адміністративний

доступ або побоювання порушення сумісності з локальними службами. У таких умовах зловмисники можуть використовувати відкриті уразливості, опубліковані в загальнодоступних репозиторіях експлойтів, для реалізації автоматизованих атак.

Додаткову небезпеку створюють відкриті порти, які залишаються активними у конфігурації пристрою без функціональної потреби. За відсутності відповідного контролю доступу порти Telnet, SSH чи HTTP дозволяють віддалене виконання команд або перегляд налаштувань без автентифікації, що значно підвищує ризик зовнішнього втручання [11].

Загрозливими є також сценарії використання IoT-пристроїв як проміжної ланки для проникнення до внутрішньої мережі навчального закладу. Якщо відсутні механізми сегментації мережі, такі як ізоляція через VLAN або фільтрація трафіку між зонами безпеки, зловмисник може через IoT-вузол отримати доступ до внутрішніх серверів, баз даних або елементів системи управління освітнім процесом.

1.2 Методи аналізу безпеки без інтеграції у систему

Оцінка інформаційної безпеки IT-систем без безпосереднього доступу до їх внутрішньої інфраструктури передбачає використання спеціалізованих методів дистанційного аналізу, які дозволяють виявити вразливості зовнішнього рівня, не модифікуючи конфігурацію або поведінку самої системи. Такі підходи особливо важливі на етапах попереднього аудиту або планування інтеграції нових компонентів, коли обмеження щодо доступу, політик або технічних умов не дозволяють використовувати повноцінні інструменти внутрішнього тестування. До цієї категорії відносять як технічні методи – зокрема пасивне мережеве спостереження, пентестування за моделлю «чорного ящика» або сканування за відкритими портами, – так і процедурні, як-от перевірку конфігурацій за відомими базами вразливостей або оцінку політик автентифікації. Детальна класифікація та порівняння таких методів наведена у табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика методів аналізу безпеки без інтеграції у систему

Метод аналізу	Принцип дії	Ключові переваги	Основні обмеження застосування
Пасивне мережеве спостереження	Моніторинг мережевого трафіку без активного втручання	Відсутність впливу на систему; виявлення незахищених протоколів та відкритих портів	Не дозволяє ідентифікувати вразливості внутрішньої логіки додатків
Зовнішнє сканування вразливостей	Аналіз відкритих сервісів на предмет відомих CVE-ідентифікаторів	Автоматизація; швидке виявлення слабких конфігурацій; мінімальний ризик впливу	Можливість хибнопозитивних результатів; обмежене охоплення внутрішніх компонент
Емуляційне (контейнерне) тестування	Відтворення поведінки підсистеми в ізольованому середовищі	Безпечність для продуктивного середовища; глибокий аналіз взаємодії компонентів	Необхідність створення адекватної моделі середовища; висока трудомісткість
Пентестування за моделлю «black-box»	Імітація зовнішнього зловмисника з обмеженим рівнем доступу	Максимальна наближеність до реального сценарію атаки; емпірична перевірка захищеності	Потребує попередньої угоди на проведення; не розкриває внутрішні політики безпеки
Перевірка за реєстрами вразливостей	Кореляція версій ПЗ з базами CVE/NVD/Exploit-DB	Швидкість; простота реалізації; підходить для великих розгортань	Не охоплює невідомі (zero-day) та специфічні для конретногосередовища вразливості [12]
Аналіз доступності зовнішніх API	Дослідження відкритих інтерфейсів, обмежень за IP, автентифікації	Виявлення грубих порушень політик доступу; неінвазивність	Не дозволяє зробити висновки про внутрішні політики обробки запитів

Отже, методи неінтрузивного аналізу безпеки забезпечують інструментальну базу для формування первинного уявлення про рівень зовнішньої захищеності ІТ-системи без ризику її дестабілізації чи потреби повного доступу до конфігурацій. Їх застосування є доцільним у фазі попередньої оцінки ризиків, формалізації вимог до захисту та планування інтеграції нових рішень. Вибір конкретного методу або їхньої комбінації має ґрунтуватися на специфіці системи, вимогах безпеки та обмеженнях, пов’язаних із доступом до інфраструктури. Університетське середовище, як об’єкт із широким спектром цифрових сервісів і підвищеним рівнем публічності, потребує саме такого обережного підходу до діагностики, що поєднує точність виявлення з мінімізацією операційного втручання. У поєднанні з подальшими активними заходами ці методи становлять основу багаторівневої моделі забезпечення інформаційної безпеки в закладах вищої освіти.

1.3 Використання машинного навчання для виявлення аномалій

У сфері забезпечення інформаційної безпеки в комп’ютерних системах університетів дедалі більшої уваги набувають інтелектуальні методи аналізу трафіку та поведінкових патернів, зокрема на основі машинного навчання. Традиційні засоби захисту, орієнтовані на сигнатурний або евристичний аналіз, виявляють лише відомі загрози та часто залишаються неефективними проти нових або модифікованих атак. Університетське ІТ-середовище, яке включає велику кількість мобільних пристроїв, дистанційних з’єднань, хмарних сервісів та IoT-компонентів, характеризується високою динамікою і непередбачуваністю, що вимагає використання адаптивних методів, здатних навчатися за даними та виявляти аномалії, які не мають чітко визначених сигнатур.

Аномалією у цьому контексті вважається будь-яка поведінка мережі або системи, що істотно відхиляється від попередньо визначеної або статистично обґрунтованої норми. Це може включати нетипову кількість з’єднань, зміну

розміру пакетів, підозрілі часові інтервали, використання нестандартних портів, відхилення у напрямках трафіку тощо. Методи машинного навчання дозволяють автоматизовано формувати модель «нормальної» поведінки системи, після чого нові спостереження порівнюються з цією моделлю, імовірнісно або класифікаційно визначаючи, чи є вони потенційно шкідливими.

Залежно від типу доступних даних та мети аналізу, застосовуються різні підходи машинного навчання:

Контрольоване навчання (англ. supervised learning), є одним із базових підходів машинного навчання, у якому метод будує узагальнену модель залежності між вхідними ознаками та наперед відомими мітками класів [13]. У задачах інформаційної безпеки комп'ютерних систем, зокрема при інтеграції нових сервісів, цей підхід застосовується для автоматизованого розпізнавання аномальної активності на основі попередньо зібраних і класифікованих прикладів мережевих подій. Модель навчається на історичних даних, де кожен запис позначено як «нормальний» або «аномальний», і після навчання здатна класифікувати нові зразки за аналогічним принципом.

У межах побудови програмного модуля для виявлення загроз в університетській мережі контрольоване навчання використовується для створення класифікатора, що розрізняє типові та нетипові (підозрілі) з'єднання. До побудови такої моделі входять етапи вибору відповідного методу, підготовки ознак, навчання на тренувальній вибірці та застосування до нових даних. Найпоширенішими методами у цьому класі є дерева рішень, метод опорних векторів, логістична регресія, наївний баєсівський класифікатор і ансамблеві методи, зокрема випадковий ліс (Random Forest) та градієнтний бустинг (FastTree).

Для узагальнення ключових характеристик найбільш релевантних методів, що застосовуються у задачах виявлення аномалій у мережевому трафіку за допомогою контрольованого навчання, складено табл. 1.3.

Таблиця 1.2 – Методи контрольованого навчання

Алгоритм	Тип алгоритму	Характеристика	Приклад використання
Logistic Regression	Лінійна класифікація	Побудова гіперплощини для розділення двох класів	Детекція фішингових URL, виявлення скомпрометованих IP
Decision Tree	Дерево рішень	Побудова дерева з бінарними умовами на ознаках	Виявлення шкідливих дій за послідовністю подій
Random Forest	Ансамбль дерев	Голосування між кількома деревами рішень	Класифікація поведінкових шаблонів у великих мережах
FastTree	Градієнтний бустинг	Послідовне уточнення помилок попередніх моделей	Виявлення складних атак зі слабко вираженими ознаками
Naive Bayes	Ймовірнісна модель	Обчислення ймовірності приналежності до класу	Визначення аномалій у трафіку DNS або SMTP
SVM (Support Vector)	Гіперплощинна модель	Максимізація межі між класами у просторах високої розмірності	Аналіз нетипових мережевих з’єднань на основі кількох параметрів

Отже, методи контрольованого навчання становлять основу для реалізації автоматизованих систем класифікації мережевих подій за наявності попередньо розмічених даних. Кожен метод має специфіку побудови моделі та по-різному реагує на вибір ознак і обсяг навчальної вибірки. У межах систем безпеки такі алгоритми застосовуються для ідентифікації потенційно небезпечних дій на основі історичних прикладів, що забезпечує адаптивність до нових атак при постійному оновленні тренувальних даних.

Навчання без вчителя (unsupervised learning) є класом методів машинного навчання, в яких модель працює з немаркованими даними, тобто без попередньо визначених класів або категорій [14]. У сфері інформаційної безпеки комп’ютерних систем, зокрема в університетських мережах, цей підхід використовується для виявлення аномальної активності у випадках, коли відсутні чіткі мітки «нормальних» і «аномальних» подій або коли нові типи

загроз ще не були задокументовані. Алгоритми такого типу аналізують структуру, щільність або відстані між об’єктами у множині ознак і виявляють ті зразки, що істотно відрізняються від більшості – тобто потенційні аномалії.

Для систематизації основних методів навчання без вчителя, що застосовуються у виявленні аномалій, подано табл. 1.3.

Таблиця 1.3 – Методи навчання без вчителя для виявлення аномалій у мережевому трафіку

Метод	Принцип дії	Характеристика	Приклад використання
K-Means	Групування даних у k кластерів за відстанню	Поділ вибірки на групи; виявлення точок, що не належать кластеру	Виявлення нестандартних з’єднань за ознаками розміру, часу, IP
DBSCAN	Кластеризація за щільністю	Визначення щільних зон; відкидання розріджених спостережень	Детекція ізольованих подій серед масового трафіку
One-Class SVM	Побудова межі для «нормального» класу	Навчання на даних однорідної природи; виявлення відхилень	Виявлення відхилень у поведінці користувача без еталонів аномалій
Isolation Forest	Ітеративне випадкове поділення даних	Швидке виявлення рідкісних і відокремлених точок	Аналіз системних логів на предмет нетипових команд
Autoencoder	Нейромережа зі стисканням і реконструкцією	Обчислення похибки відновлення	Виявлення атак у потокових даних з великою кількістю ознак

Отже, методи навчання без вчителя дозволяють виявляти нетипову активність у мережевому середовищі без потреби у попередньому маркуванні подій, що робить їх корисними в ситуаціях з високою варіативністю даних або невизначеністю класів. Вони формують математичну модель «типової» поведінки й автоматично виокремлюють потенційні відхилення, що можуть свідчити про вторгнення або зловживання доступом. Універсальність підходів кластеризації, статистичного аналізу та глибокого навчання дозволяє гнучко

адаптувати їх до специфіки академічних мереж, враховуючи сезонність, активність студентів і динаміку навантажень.

У якості вхідних даних для навчання моделі можуть використовуватись як статичні лог-файли (у форматі CSV, NetFlow, PCAP тощо), так і потокові дані, що надходять у режимі реального часу [15]. Типовими ознаками для аналізу є: кількість пакетів, тривалість з'єднання, об'єм переданих даних, час між запитами, напрямок руху, тип протоколу тощо. При цьому критично важливо забезпечити коректну нормалізацію даних, обробку пропусків, перетворення категоріальних ознак (наприклад, хешування або one-hot encoding) та розподіл вибірки на тренувальну й тестову частини для об'єктивної оцінки якості моделі.

Результати застосування методів машинного навчання оцінюються за допомогою стандартних метрик, таких як точність, повнота, специфічність, середньозважена F-міра та площа під кривою ROC. Особливої уваги потребує аналіз помилок другого роду (false negatives), оскільки недетектовані аномалії можуть свідчити про прогалини у захисті.

Застосування методів машинного навчання у виявленні аномалій створює можливості для гнучкого, автоматизованого захисту інформаційної інфраструктури університету, який адаптується до змін у поведінці користувачів та мережевих умов. Використання таких підходів у складі автономних модулів дозволяє не лише знизити навантаження на IT-персонал, а й значно скоротити час реагування на потенційні інциденти, що є критично важливим у сучасних умовах постійних кіберзагроз.

1.4 Вибір інструментальних засобів реалізації

У процесі розроблення програмного модуля для виявлення аномальної активності в університетських комп'ютерних системах ключову роль відіграє обґрунтований вибір інструментальних засобів, що забезпечують не лише технічну реалізацію, але й сумісність з існуючою IT-інфраструктурою. Зважаючи на специфіку завдання – обробку логів і трафіку, інтеграцію

алгоритмів машинного навчання, реалізацію графічного інтерфейсу користувача та взаємодію з локальними системами – доцільно розглянути найбільш поширені мови програмування та технології, що підтримують зазначені функціональні можливості. Серед універсальних і широко застосовуваних мов для створення безпекових рішень на основі машинного навчання виділяються Python, Java та C#, кожна з яких має свою область застосування в задачах аналізу трафіку та безпечної інтеграції систем.

Python є інтерпретованою мовою високого рівня, яка набула широкого визнання у сфері наукових обчислень і машинного навчання завдяки бібліотекам NumPy, pandas, scikit-learn і TensorFlow [16]. Вона дозволяє швидко створювати прототипи моделей та обробляти великі обсяги даних у форматах CSV, PCAP або JSON. Python особливо зручний для реалізації серверної логіки без графічного інтерфейсу або у зв’язці з Jupyter-ноутбуками для аналітики.

Java є строго типізованою мовою з потужною екосистемою, що активно використовується в корпоративних IT-рішеннях та системах з високими вимогами до стабільності [17]. Завдяки платформонезалежності та вбудованим засобам для багатопотоковості Java застосовується для побудови модулів безпеки в розподілених або хмарних середовищах. Наявність бібліотек типу Weka, Deeplearning4j і Apache Spark MLlib дозволяє створювати модулі машинного навчання з підтримкою кластерних обчислень.

C# – об’єктно-орієнтована мова програмування, розроблена компанією Microsoft, яка ідеально підходить для побудови десктопних застосунків з графічним інтерфейсом [18]. Завдяки бібліотеці ML.NET C# дозволяє інтегрувати моделі машинного навчання безпосередньо в середовище .NET, що робить її зручною для реалізації систем моніторингу, які працюють локально на клієнтських машинах. Її застосування особливо ефективне у випадках, коли система безпеки має бути вбудована в існуючі Windows-орієнтовані сервіси.

З огляду на широкий спектр можливостей кожної з розглянутих мов програмування, доцільно здійснити їх порівняльний аналіз за ключовими характеристиками, які мають визначальне значення для побудови програмного

модуля в системі забезпечення безпеки комп'ютерної інфраструктури університету (табл. 1.4).

Таблиця 1.4 – Порівняння мов програмування

Характеристика	Python	Java	C#
Синтаксис	Дуже лаконічний	Строго формалізований	Сучасний та гнучкий
Підтримка GUI	Посередня (tkinter, PyQt)	Обмежена (JavaFX, Swing)	Висока (WinForms, WPF, MAUI)
Швидкість виконання (середня, ms)	120–180	90–130	70–100
Ресурсоємність	Висока при обробці GUI	Середня	Помірна
Можливість роботи з ML	Висока (scikit-learn, TensorFlow)	Середня (Weka, DL4J)	Висока (ML.NET)
Документація та підтримка	Широка	Широка	Офіційна MSDN та інтегровані засоби
Інтеграція з Windows-системами	Обмежена	Обмежена	Повна (через .NET)
Підтримка локалізованих UI	Часткова	Обмежена	Повна (WinForms, WPF)

Виходячи з порівняльного аналізу, мова програмування C# виявляється оптимальним вибором для реалізації програмного модуля системи забезпечення безпеки в університетському середовищі. Вона поєднує у собі високу продуктивність виконання, ефективну підтримку графічного інтерфейсу користувача та глибоку інтеграцію з операційною системою Windows, яка є типовим середовищем для більшості адміністративних та навчальних IT-систем. Завдяки платформі .NET забезпечується повноцінна підтримка роботи з базами даних, локальними службами, системними журналами та ресурсами, що критично важливо для безпечної обробки подій. Крім того, використання бібліотеки ML.NET дозволяє реалізувати вбудовані механізми машинного навчання без потреби у сторонніх обгортках або мовах. Таким чином, вибір C#

забезпечує збалансовану технічну основу для розробки надійного, продуктивного та функціонально повного застосунку.

З огляду на вибір мови програмування C# як основної технологічної платформи для реалізації програмного модуля системи забезпечення безпеки, наступним кроком є обґрунтування вибору відповідного середовища розробки. Серед найбільш поширених та функціонально придатних IDE для роботи з C# слід виділити Visual Studio 2022, Visual Studio Code та JetBrains Rider. Кожне з цих середовищ має власну спеціалізацію, архітектурні особливості та ступінь інтеграції з технологіями .NET, що дозволяє адаптувати вибір відповідно до конкретних вимог розробника – від створення повноцінних десктопних інтерфейсів до кросплатформених CLI-інструментів.

Visual Studio 2022 – це повнофункціональне інтегроване середовище розробки від Microsoft, яке надає повний стек засобів для створення, налагодження, тестування та розгортання застосунків на платформі .NET [19]. Воно підтримує проєкти WinForms, WPF, ASP.NET, має вбудовані інструменти роботи з базами даних, інтерфейсом ML.NET та модулем NuGet. Середовище забезпечує найглибшу інтеграцію з Windows API, що особливо важливо для розробки десктопних систем безпеки з локальною взаємодією.

Visual Studio Code є полегшеним кросплатформеним редактором коду, який підтримує C# за допомогою розширень, зокрема C# Dev Kit і Omnisharp [20]. Хоча VS Code не має повноцінного дизайнера форм для WinForms, він є зручним для створення консольних і серверних частин застосунків, написання логіки модулів, а також обробки логів і даних у супутніх скриптах. Його перевагою є швидкодія, розширюваність і зручність при роботі з Git та JSON-конфігураціями.

JetBrains Rider – це потужне середовище розробки на основі ReSharper та платформи IntelliJ, що підтримує C#, .NET, ASP.NET Core, а також Unity. Rider орієнтований на розробників, які працюють у мультиплатформеному середовищі, і дозволяє інтегруватися з Docker, Azure, базами даних та системами контролю версій [21]. Незважаючи на стороннє походження, Rider

пропонує продуктивні засоби рефакторингу, аналізу коду та підтримку XAML для WPF-застосунків.

Для побудови прикладного модуля безпеки, орієнтованого на платформу C#/.NET, важливо враховувати не лише підтримку мови, а й такі характеристики, як робота з графічним інтерфейсом, вбудовані інструменти аналітики, налагодження, інтеграція з ML.NET, керування залежностями, підтримка систем контролю версій та взаємодія з базами даних. З огляду на це, доцільно провести порівняння трьох ключових середовищ – Visual Studio 2022, Visual Studio Code та JetBrains Rider – за релевантними критеріями, наведеними у табл. 1.5.

Таблиця 1.5 – Порівняльна характеристика середовищ розробки

Характеристика	Visual Studio 2022	Visual Studio Code	JetBrains Rider
Підтримка .NET та .NET Core	Повна	Через розширення (C# Dev Kit)	Повна
Підтримка WinForms/WPF Designer	Повна	Відсутня	Обмежена (через XAML-редактор)
Підтримка ML.NET	Інтегрована	Часткова (через CLI)	Часткова (через SDK-інтеграцію)
Налагодження (debugging)	Повне налагодження з GUI	Легке, але базове	Потужне, із глибоким аналізом
Швидкодія при запуску IDE	Відносно висока	Висока	Висока
Інтеграція з Git	Вбудована	Вбудована	Вбудована
Робота з NuGet	Повна підтримка	Через командний рядок	Повна підтримка
Підтримка баз даних	Інтегровані SQL Server Tools	Через розширення	Вбудований Data Viewer + SQL IDE
Підтримка XAML	Повна	Відсутня	Часткова
Цільова аудиторія	Корпоративна, розширена GUI	Легка CLI-розробка, кросплатформені	Кросплатформені .NET-розробники

Порівняльний аналіз середовищ розробки засвідчує, що Visual Studio 2022 є найбільш повноцінним інструментом для реалізації графічного C#-застосунку з підтримкою WinForms, ML.NET та глибокою інтеграцією з платформою Windows. Visual Studio Code, хоча й має обмежену підтримку GUI, є ефективним вибором для розробки CLI-компонентів, скриптів і допоміжних утиліт. JetBrains Rider, своєю чергою, надає розширену підтримку для мультиплатформеного .NET-розроблення та підходить для проєктів, де важлива продуктивність, модульність і інтеграція з сучасними DevOps-практиками.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Побудова блок-схем алгоритмів програмного забезпечення

У процесі розроблення програмного модуля системи забезпечення інформаційної безпеки важливу роль відіграє чітке формалізоване представлення логіки роботи основних функціональних компонентів. Побудова блок-схем дозволяє відобразити послідовність виконання операцій, розгалуження, перевірку умов та взаємодію між модулями, що суттєво полегшує як реалізацію, так і подальший супровід програмного забезпечення.

На рис. 2.1 наведено блок-схему алгоритму тренування моделі, що реалізована у відповідному функціональному модулі системи.

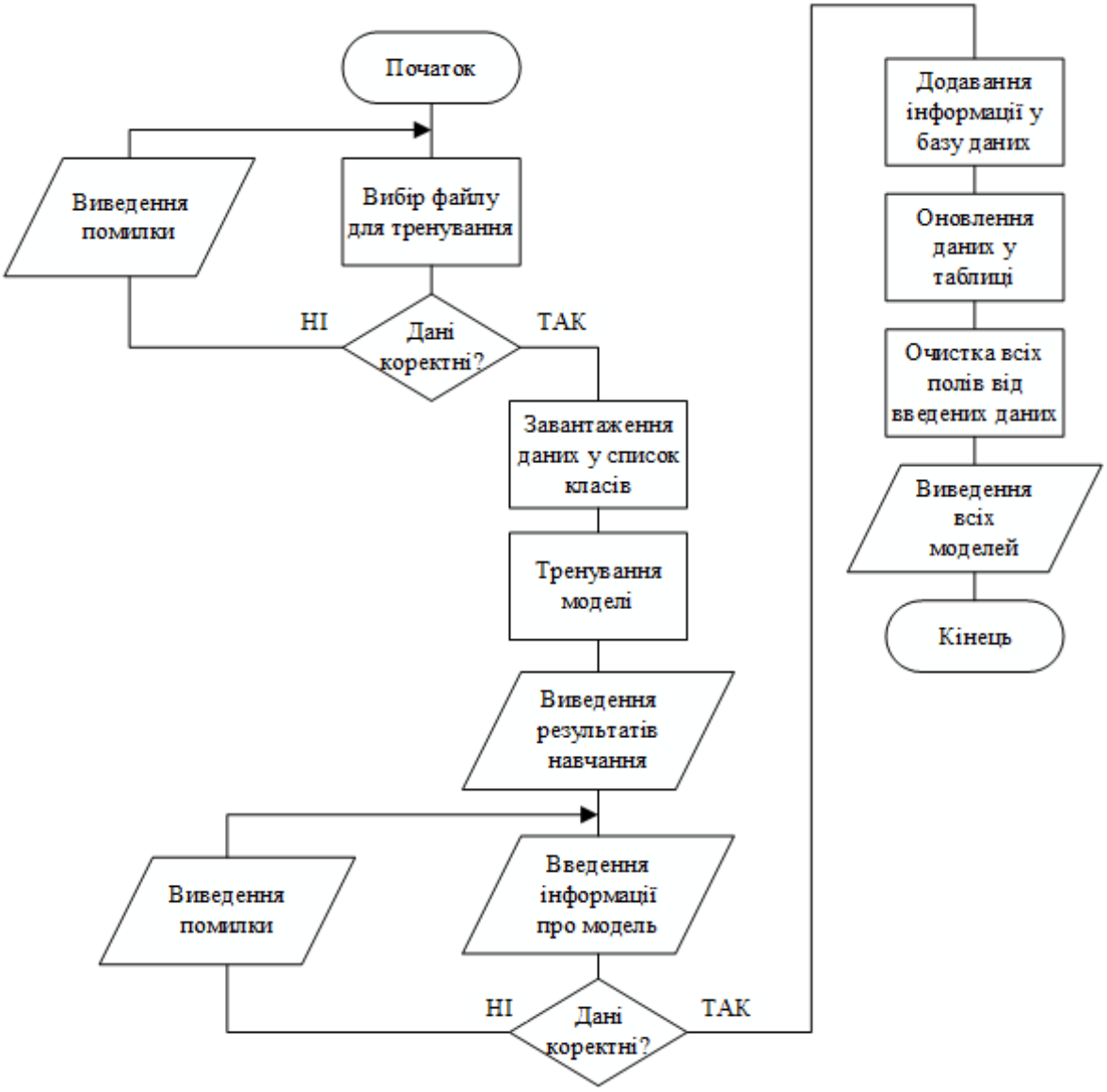


Рисунок 2.1 – Алгоритм тренування моделі

Алгоритм розпочинається з ініціалізації процесу, де користувач взаємодіє з інтерфейсом програми з метою вибору файлу для тренування. Це, як правило, CSV-файл або інший структурований формат, що містить вхідні дані для навчання моделі машинного навчання. Далі запускається етап перевірки коректності даних – система аналізує наявність необхідних стовпців, формат числових та категоріальних значень, відсутність пропусків, дублювання або конфліктів у мітках класів. Якщо дані не відповідають вимогам, користувач отримує повідомлення про помилку, і алгоритм повертається у вихідний стан.

У випадку, коли дані коректні, відбувається завантаження даних у список класів, де кожен рядок перетворюється на об’єкт відповідної структури, що буде використано як вхідний приклад для моделі. Після успішного завантаження запускається етап тренування моделі, що виконується із використанням попередньо визначеного алгоритму машинного навчання. Навчання супроводжується побудовою конвеєра обробки, тренуванням та оцінюванням моделі на вбудованому тестовому наборі.

Після завершення навчання система переходить до виведення результатів навчання, які можуть включати точність класифікації, матрицю помилок, AUC, F1-метрику тощо. Ці дані автоматично виводяться у текстове поле або іншу частину GUI, доступну користувачу.

На основі отриманих результатів користувач переходить до введення інформації про модель. Система виконує перевірку введених даних на коректність – порожні поля, некоректні символи, дублікати назв. Якщо помилка виявлена, виводиться відповідне повідомлення і повернення до етапу вводу.

Якщо ж дані відповідають вимогам, запускається завершальна послідовність операцій, яка включає:

– додавання інформації про модель у базу даних;

– оновлення вмісту таблиці моделей;

– очищення всіх полів вводу, щоб забезпечити підготовку до наступного циклу;

– виведення усіх моделей, що наразі збережені в системі, включаючи щойно додану.

Після виконання всіх дій алгоритм коректно завершує свою роботу.

Алгоритм виявлення аномалії, поданий на рис. 2.2, описує логіку роботи модуля моніторингу мережевої активності з використанням попередньо натренованої моделі машинного навчання. Його побудовано з урахуванням потреби в безперервному зборі трафіку, передачі даних у модель, отриманні прогнозу та динамічному керуванні процесом моніторингу. Послідовність кроків забезпечує як повну ініціалізацію системи, так і обробку виключних ситуацій, що дозволяє підтримувати стабільну роботу застосунку навіть у разі відсутності моделі чи збоїв в обміні.

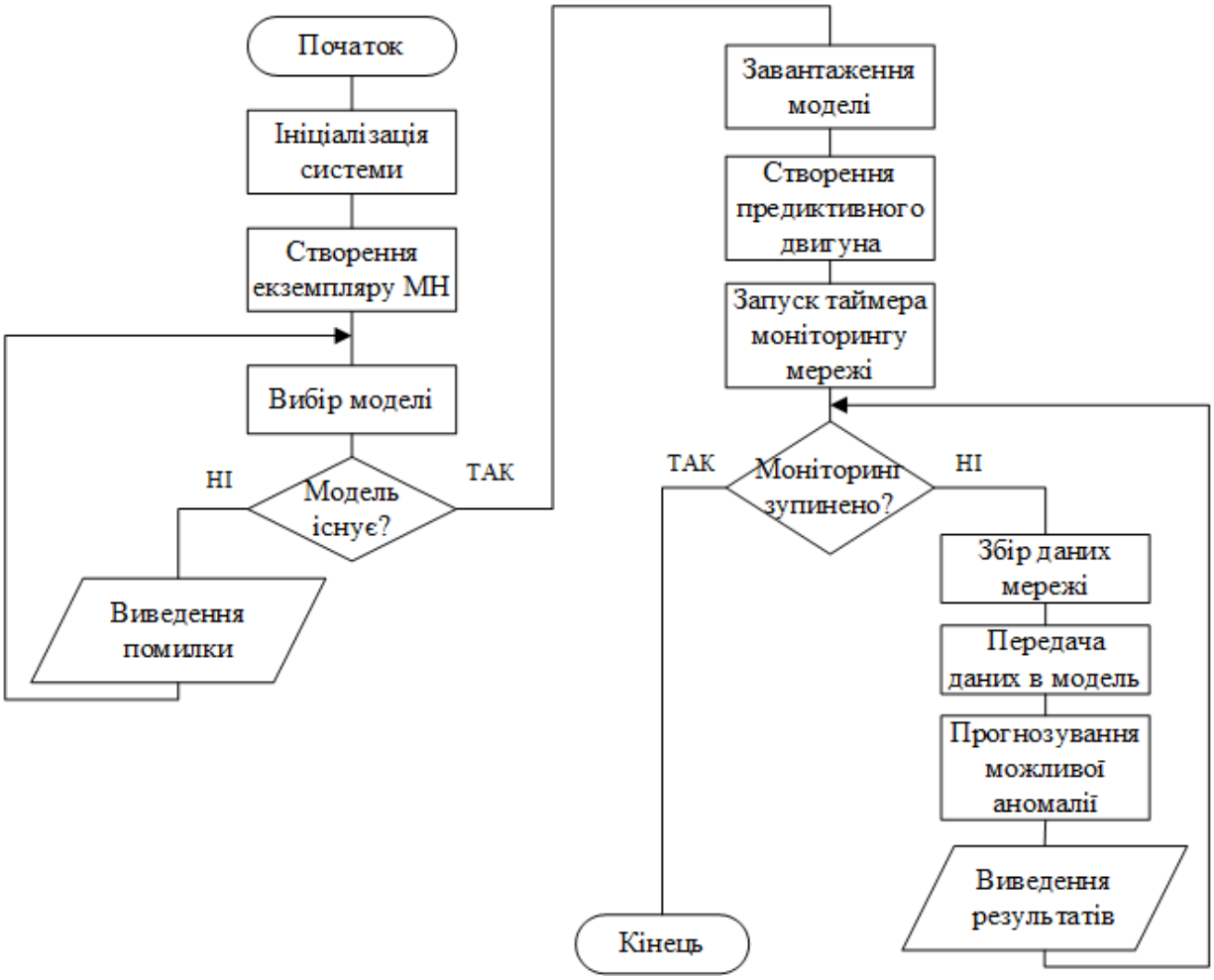


Рисунок 2.2 – Алгоритм виявлення аномалії

Алгоритм починається з етапу ініціалізації системи, в ході якого завантажуються базові налаштування, конфігураційні параметри або

залежності. Далі створюється екземпляр класу машинного навчання, тобто об’єкт, що відповідає за виконання інференсу (прогнозу) на основі обраної моделі. Після цього користувач виконує вибір моделі – наприклад, з локального списку збережених моделей або з бази даних. Якщо модель відсутня або вибрана помилково, система повідомляє про це й припиняє виконання, уникаючи подальших помилок. Якщо ж модель наявна, виконується її завантаження у пам’ять. Далі створюється предиктивний двигун, який відповідає за передачу даних у модель і отримання відповіді у реальному часі. Одразу після цього запускається таймер моніторингу, що активує регулярну перевірку стану мережі з заданим інтервалом.

На кожному циклі перевіряється, чи моніторинг зупинено – якщо так, виконання припиняється, і система переходить у завершальний стан. Якщо ж моніторинг продовжується, відбувається збір актуальних даних мережі, що можуть включати розмір пакетів, IP-адреси, час передачі, кількість підключень тощо. Ці дані передаються до моделі, де відбувається прогнозування ймовірної аномалії на основі навченої логіки класифікації.

У результаті виконання передбачення користувачу виводиться інформація про результат, зокрема повідомлення про виявлену підозрілу активність, її індикатори або ступінь ймовірності. Цикл повторюється до тих пір, поки не буде ініційовано завершення моніторингу.

Даний алгоритм дозволяє вбудовано реалізувати механізм виявлення мережевих аномалій у реальному часі з автоматизованою реакцією на вхідні події, що відповідає вимогам до сучасних засобів інформаційної безпеки в інституційних IT-середовищах.

2.2 Діаграми архітектури системи

У процесі розроблення програмного модуля системи забезпечення безпеки при інтеграції нових технологій до комп’ютерної інфраструктури університету критично важливо сформувати узгоджену архітектурну модель, яка б забезпечувала надійність, масштабованість та підтримуваність

реалізованого рішення. Побудова відповідної архітектури дозволяє структуровано розподілити логіку системи, чітко відокремити прикладну частину від інтерфейсу користувача та механізмів збереження даних, що істотно спрощує процеси налагодження, супроводу та подальшого розширення функціональності. Враховуючи функціональні вимоги системи, наявність графічного інтерфейсу, роботу з локальною базою даних і використання моделей машинного навчання, доцільним є застосування трьохрівневої архітектури як найбільш раціонального рішення.

Трьохрівнева архітектура передбачає поділ програмної системи на три логічно незалежні рівні: рівень презентації (Presentation Layer), рівень прикладної логіки (Business Logic Layer) та рівень доступу до даних (Data Access Layer). Такий підхід забезпечує модульність коду, дозволяє оновлювати або змінювати компоненти одного рівня без порушення функціонування інших, а також сприяє дотриманню принципів інкапсуляції та розмежування відповідальностей. Для даної системи це особливо важливо з огляду на потребу в окремому обслуговуванні: інтерфейсу користувача, логіки роботи з моделями машинного навчання, а також збереження результатів і метаінформації у базі даних. Саме тому трьохрівнева архітектура є обґрунтованим вибором для досягнення структурної чіткості та надійності у функціонуванні програмного забезпечення.

У межах трьохрівневої архітектури програмної системи окрему роль відіграє рівень доступу до даних, який відповідає за взаємодію з базою даних, інкапсуляцію SQL-запитів та забезпечення незалежності прикладної логіки від способу зберігання інформації. Відповідно до цього принципу, всі операції читання, запису, оновлення та видалення даних реалізуються через спеціалізовані класи-постачальники (провайдери), які уніфікують доступ до відповідних сутностей. Для наочності структура цього рівня представлена у вигляді діаграми класів на рис. 2.3.

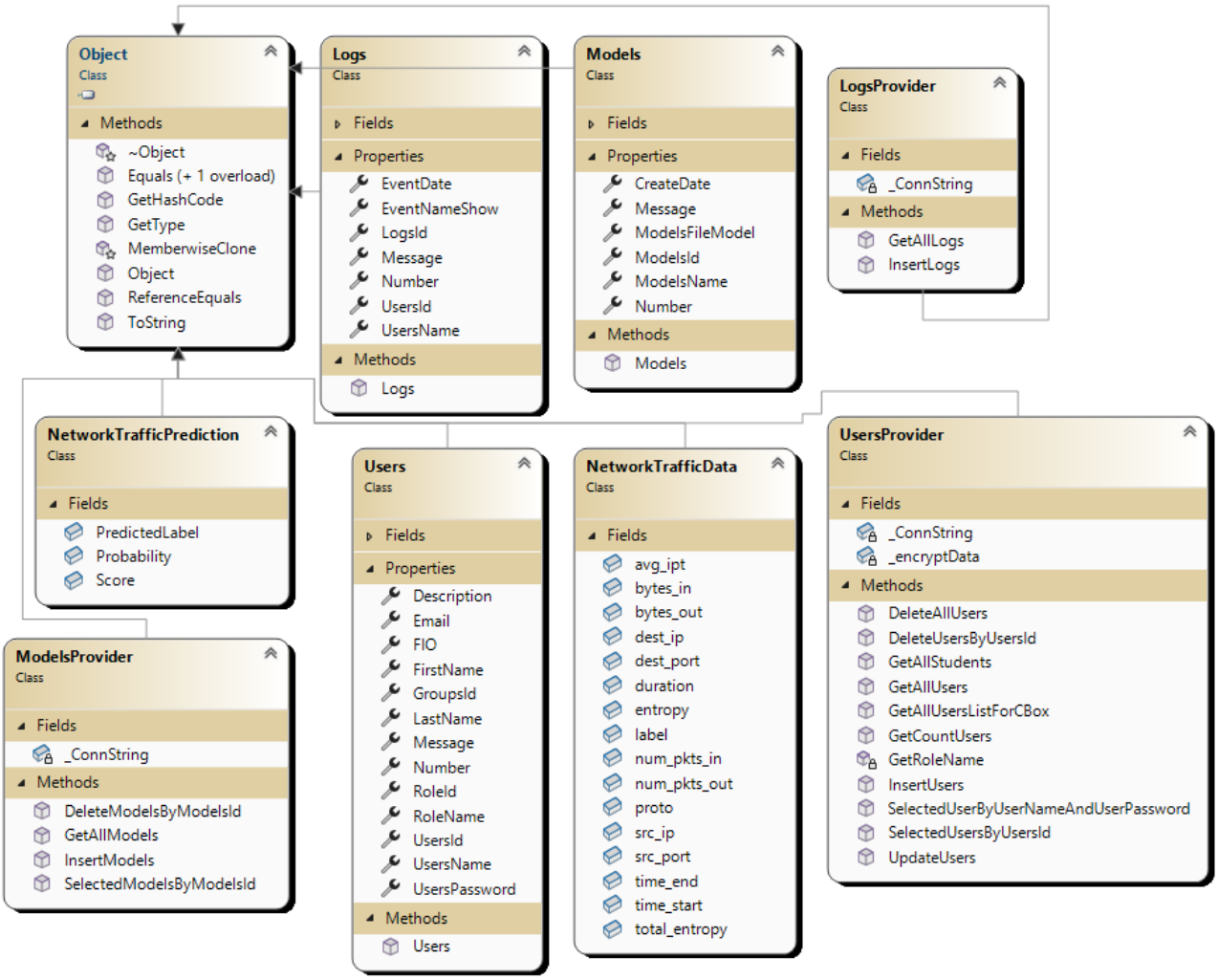


Рисунок 2.3 – Діаграма класів рівня доступу до даних

Діаграма класів рівня даних складається з:

– класу ModelsProvider, який реалізує доступ до інформації про навчальні моделі. Він містить методи для додавання нових моделей, видалення моделей за ідентифікатором, отримання повного списку моделей, а також вибірки моделей за визначеними критеріями;

– класу LogsProvider, що відповідає за обробку журналів подій системи. Він включає методи для отримання всіх записів журналу та вставки нових логів, які фіксують ключові дії користувача або внутрішні події модуля безпеки;

– класу UsersProvider, який реалізує логіку взаємодії з таблицею користувачів. Його функціонал охоплює повний набір CRUD-операцій: додавання, оновлення, видалення користувачів, а також пошук за іменем,

ідентифікатором, роллю або іншими атрибутами. Крім того, реалізована можливість формування списків для компонентів інтерфейсу;

– класу NetworkTrafficData, що відповідає за структуру збереження мережевих характеристик для подальшого аналізу та передачі у модель машинного навчання. Клас містить поля, які відображають усі ключові параметри мережевого трафіку, включаючи кількість пакетів, напрямки, розміри, порти, протоколи, ентропію та часові мітки;

– класу NetworkTrafficPrediction, який описує структуру результату прогнозування. Він містить такі поля, як передбачений клас (аномалія або норма), ймовірність та оцінка моделі (score), що повертаються після застосування моделі машинного навчання;

– класу Users, який є представленням користувача системи з усіма пов’язаними властивостями: ім’я, прізвище, email, роль, логін, пароль тощо. Він використовується як структурний елемент для передачі даних між рівнями;

– класу Models, що відображає метаінформацію про збережені моделі: їхню назву, дату створення, опис, тип, унікальний ідентифікатор і шлях до відповідного файлу.

– класу Logs, який структурує записи журналу: дата події, ім’я події, повідомлення, користувач і пов’язані ідентифікатори. Цей клас дозволяє фіксувати як системні дії, так і дії користувача в інтерфейсі.

Таким чином, представлена діаграма класів демонструє чітке розмежування відповідальностей між компонентами, що дозволяє забезпечити незалежність рівнів, розширюваність архітектури та контроль над обміном даними в межах системи.

Для реалізації функціональної взаємодії користувача з програмною системою, що забезпечує виявлення аномалій у мережевому середовищі університету, було спроєктовано та реалізовано окремий рівень користувацького інтерфейсу. Основу цього рівня становлять форми, які безпосередньо відображають структуру модулів системи та забезпечують реалізацію взаємодії з користувачем через графічні компоненти. Візуальна

структура цього рівня наведена на рис. 2.4 у вигляді діаграми класів, де кожен клас є формою, успадкованою від базового класу Form.

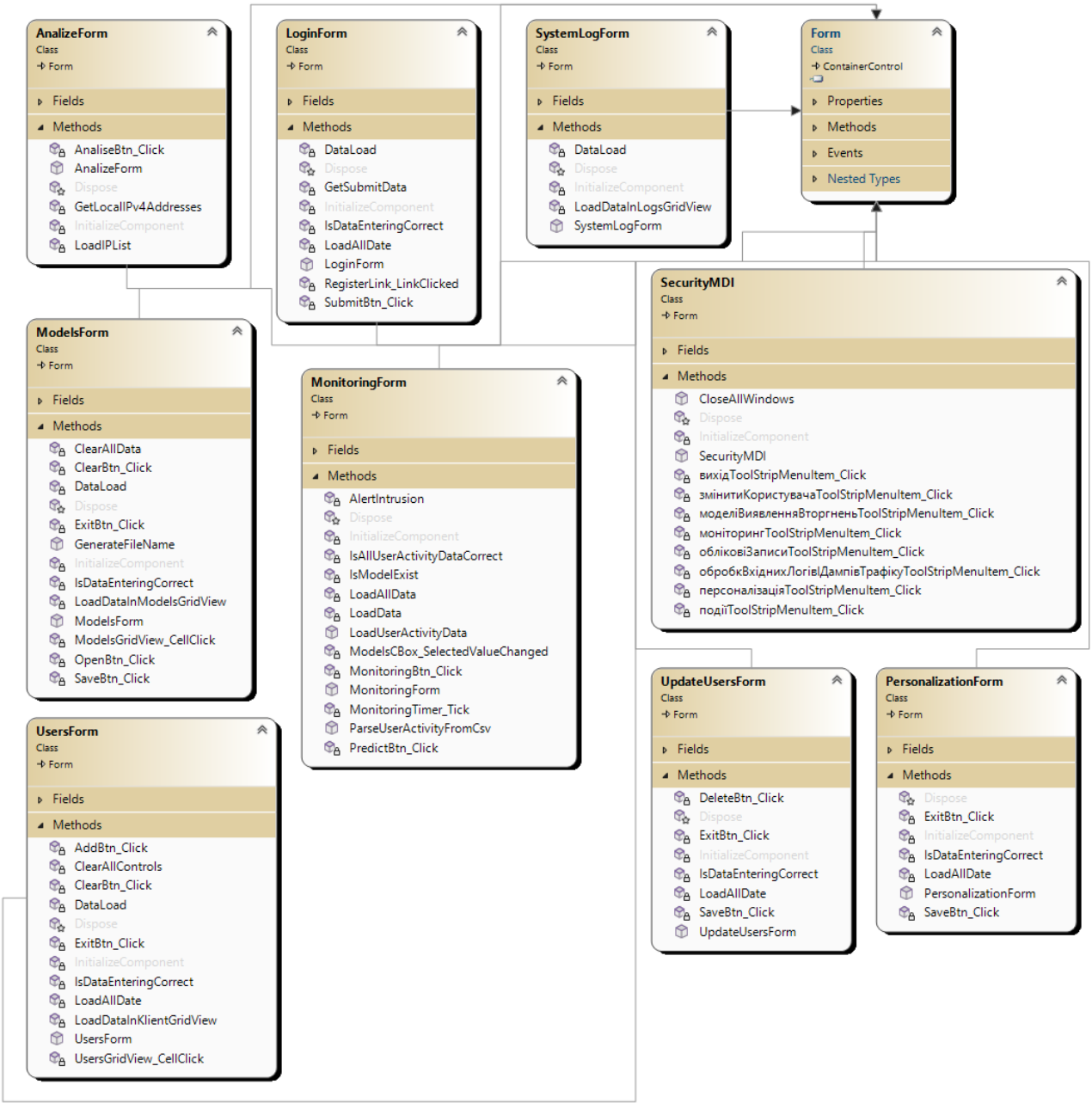


Рисунок 2.4 – Діаграма класів рівня користувацького інтерфейсу системи

Діаграма класів рівня інтерфейсу користувача складається з:

– класу SecurityMDI, який є головною контейнерною формою (Multiple Document Interface), через яку реалізується навігація між модулями системи;

– класу LoginForm, що відповідає за процес авторизації користувача перед входом у систему. У формі реалізовані методи завантаження списку

користувачів, перевірки облікових даних та реєстрації нового користувача через подію RegisterLinkClicked;

– класу UsersForm, який забезпечує інтерфейс для управління користувачами системи. Він містить методи додавання, редагування, видалення, очищення полів, а також виведення списку користувачів у таблицю UsersGridView;

– класу UpdateUsersForm, що надає функціонал редагування вже існуючих користувачів. До основних методів відносяться завантаження обраного користувача, перевірка введених даних та збереження змін;

– класу PersonalizationForm, який відповідає за налаштування персональних даних користувача. Він містить механізми валідації введених параметрів, а також методи для завантаження та збереження персональної інформації;

– класу ModelsForm, який реалізує інтерфейс для управління моделями машинного навчання. Форма дозволяє додавати нові моделі, завантажувати їх із файлу, очищувати поля, відображати таблицю всіх моделей та зберігати інформацію в базу даних;

– класу MonitoringForm, що є основним інструментом для активного моніторингу мережевої активності. У класі реалізовано функції запуску таймера, зчитування мережевих даних, передбачення аномалій, виведення результатів і завантаження даних для аналізу;

– класу SystemLogForm, який забезпечує перегляд системних подій та логів, що автоматично фіксуються під час роботи програми;

– класу AnalyzeForm, що надає інтерфейс для додаткового аналізу мережевого трафіку, зокрема локального аналізу IP-адрес.

Рівень інтерфейсу користувача побудований на основі модульного підходу, де кожна форма виконує чітко визначену роль у загальній структурі системи. Використання шаблону MDI дозволяє централізовано керувати всіма підлеглими формами, а розмежування функцій між різними вікнами забезпечує логічну ізоляцію відповідальностей. Уся взаємодія реалізується через подійно-

орієнтовану модель, що дозволяє забезпечити інтерактивність, зручність у користуванні та легкість у підтримці інтерфейсу. Такий підхід повністю відповідає вимогам до розробки сучасного захищеного прикладного програмного забезпечення з функціями аналітики.

Відповідно до концепції трьохрівневої архітектури, рівень бізнес-логіки виконує ключову роль у реалізації прикладних алгоритмів, перевірки коректності введених даних, обробки трафіку, а також забезпечення взаємодії між інтерфейсом користувача та рівнем даних. Саме на цьому рівні реалізуються незалежні від інтерфейсу механізми прийняття рішень, валідації, шифрування та аналітики. Його структура наведена на рис. 2.5 у вигляді діаграми класів.

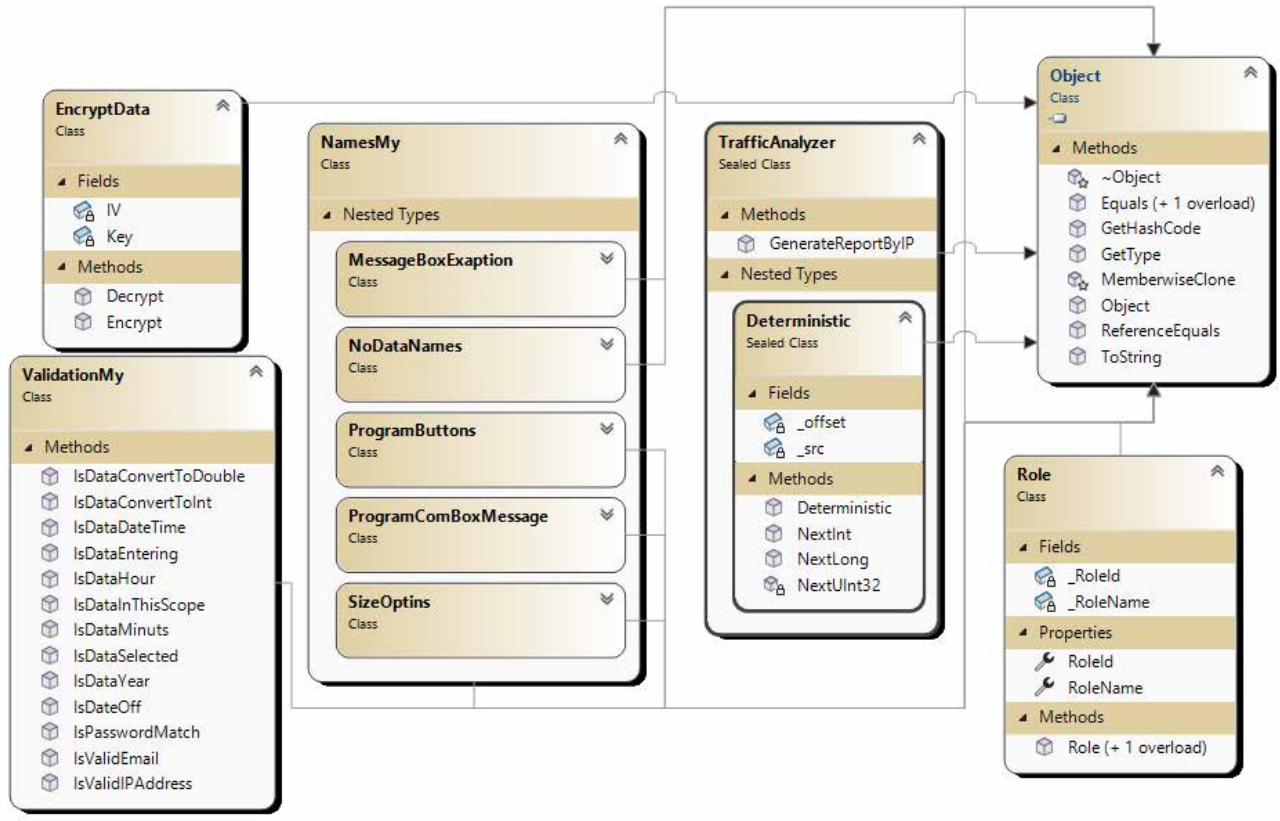


Рисунок 2.5 – Діаграма класів рівня бізнес-логіки системи

Діаграма класів рівня бізнес-логіки складається з:

– класу ValidationMy, який реалізує набір методів для перевірки коректності вхідних даних;

– класу EncryptData, що виконує функції шифрування та дешифрування даних. Він містить поля IV (ініціалізаційний вектор) та Key (ключ

шифрування), а також методи Encrypt і Decrypt, які реалізують відповідні криптографічні операції. Клас використовується для захисту паролів користувачів або інших чутливих даних у базі;

– класу TrafficAnalyzer, який відповідає за побудову аналітичних звітів на основі IP-адрес. Основний метод GenerateReportByIP виконує агрегацію та аналіз трафіку відповідно до заданих фільтрів або критеріїв. Внутрішня структура класу розширюється вкладеним класом Deterministic;

– вкладеного класу Deterministic, що використовується в контексті генерації контрольованих псевдовипадкових значень. Він містить поля _offset та _src, а також методи NextInt, NextLong, NextUInt32, які дозволяють відтворювано формувати послідовності на основі певного джерела. Це може використовуватися для генерації тестових даних або симуляції подій;

– класу NamesMy, який виконує роль контейнера для вкладених типів, що структуровано зберігають назви кнопок, повідомлень, розмірів, винятків і відсутніх полів;

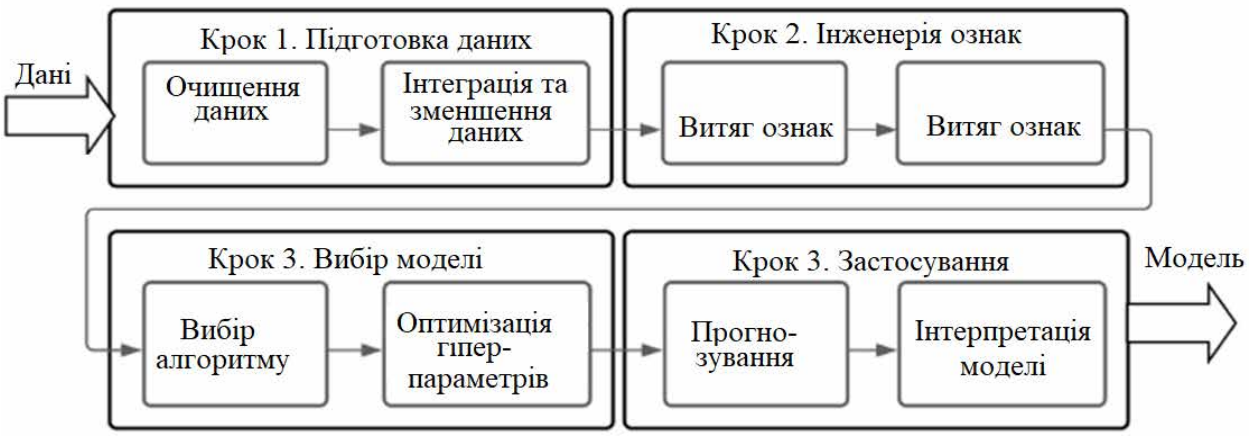
– класу Role, що представляє логічну модель ролей користувача. Він містить поля _RoleId, _RoleName та відповідні властивості, які використовуються для призначення прав доступу та побудови ролевої моделі безпеки. Клас дозволяє структурувати права доступу до частин системи на рівні бізнес-логіки.

Наведена діаграма ілюструє гнучку та ізольовану реалізацію ключових прикладних компонентів, які забезпечують логічну обробку даних, захист інформації, валідацію вводу, генерацію аналітики та контроль доступу. Такий підхід відповідає принципам розділення відповідальностей і дозволяє централізовано управляти правилами системної поведінки без залежності від візуального або структурного шару. У сукупності ці класи формують ядро прикладної логіки, що забезпечує інтелектуальність та безпечність системи.

2.3 Розробка концепції та математичної моделі системи машинного навчання

Побудова системи виявлення аномалій у мережевому трафіку з використанням алгоритмів машинного навчання вимагає не лише реалізації програмного коду, але й формалізованого підходу до визначення структури навчального процесу, типів вхідних даних, методу прогнозування та логіки використання моделі у продуктивному середовищі. Розробка концепції такої системи передбачає чітке визначення етапів обробки даних – від первинного збору до отримання передбачень – з урахуванням циклічності, адаптивності та наявності користувацького впливу. У межах цього підходу формується логічна модель побудови моделі машинного навчання, яка слугує основою для розробки програмного забезпечення, здатного до самонавчання, масштабування та автоматичного реагування на аномальні події.

Для наочного представлення процесу функціонування системи було реалізовано узагальнену схему побудови та використання моделі машинного навчання, яка висвітлена на рис. 2.6.



Римунок 2.6 – Структура побудови та використання моделі

Схема ілюструє два основні етапи життєвого циклу моделі: етап навчання та етап використання. Першим етапом є підготовка та завантаження вхідного датасету, який складається з набору параметрів мережевого трафіку (наприклад, кількість байтів, кількість пакетів, протоколи, порти, ентропія тощо). Ці дані проходять етап попередньої обробки, що включає нормалізацію,

заповнення пропусків, перетворення категоріальних ознак у числові, а також формування структури, сумісної з тренувальним конвеєром.

Наступним етапом є тренування моделі з використанням вибраного алгоритму. Після завершення навчання модель зберігається в серіалізованому вигляді (файл .zip або .bin), що дозволяє повторно її завантажувати без потреби у повторному тренуванні. Це завершує фазу навчання.

Під час продуктивного використання система виконує збір поточних мережевих характеристик, що також проходять блок попередньої обробки – із використанням тих самих трансформацій, що були застосовані до навчальних даних. Потім дані передаються у предиктивний двигун, який виконує інференс на основі завантаженої моделі. На виході формується прогноз: ознака класу (норма / аномалія) та додатково – ймовірність або оцінка довіри до передбачення (score).

Розглядаючи побудову математичної моделі для виявлення аномалій у комп'ютерній мережі, важливо передусім формалізувати вхідні дані, які є основою процесу навчання та прогнозування.

У контексті розв'язуваної задачі, кожен запис трафіку (або потоку) представляється вектором ознак, який включає як числові параметри, що описують кількісні характеристики з'єднання, так і категоріальні, які фіксують адресну та протокольну інформацію. Цей вектор підлягає подальшій обробці, трансформації та подачі на вхід класифікаційної моделі.

Нехай $D = \{(x_i, y_i)\}_{i=1}^{N}$ – навчальна вибірка з N об'єктів, де x_i – вектор ознак, а $y_i \in \{0,1\}$ – бінарна мітка, що вказує на належність до нормального або аномального класу відповідно.

Початковий вектор ознак має вигляд:

$$x_i^{(0)} = (x_{i1}, x_{i2}, x_{i3}, x_{i4}, x_{i5}, x_{i6}, x_{i7}x_{i8}, c_{i1}, c_{i2}, c_{i3}, c_{i4}, c_{i5},)^T , \qquad (2.1)$$

де числові компоненти інтерпретуються як:

x_{i1} – середній інтервал між пакетами (с);

x_{i2} – обсяг вхідного трафіку (байт);

x_{i3} – обсяг вихідного трафіку (байт);

x_{i4} – ентропія потоку;

x_{i5} – кількість вихідних пакетів;

x_{i6} – кількість вхідних пакетів;

x_{i7} – сумарна ентропія;

x_{i8}– тривалість сесії (с).

Категоріальні компоненти:

c_{i1} – протокол передачі (TCP, UDP тощо);

c_{i2} – IP-адреса призначення;

c_{i3} – порт призначення;

c_{i4} – IP-адреса джерела;

c_{i5} – порт джерела.

З метою обробки векторів ознак перед подачею в модель здійснюється перетворення категоріальних змінних у числову форму. Для цього використовується одновимірне бінарне кодування (One-Hot Encoding) для протоколу c_{i1}, що створює вектор:

$$\phi(c_{i1}) \in \{0,1\}^{K_1} . \quad (2.2)$$

Хешоване бінарне кодування (One-Hot Hash Encoding) для IP-адрес та портів, тобто:

$$c_{ij} \mapsto \phi(c_{ij}) \in \{0,1\}^{K_j}, j = 2, \dots, 5 . \quad (2.3)$$

У результаті отримується розширений вектор ознак, який поєднує нормалізовані числові та закодовані категоріальні компоненти.

Числові ознаки x_{ij} підлягають стандартизації за формулами:

$$\bar{x}_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j} . \quad (2.4)$$

Це дозволяє привести всі числові ознаки до однакового масштабу, що є критично важливим для ефективної роботи більшості моделей машинного навчання.

Об’єднавши нормалізовані числові та категоріальні ознаки, отримується:

$$x_i = (\bar{x}_{i1}, \bar{x}_{i2}, \dots, \bar{x}_{i8}, \phi(c_{i1})^T, \dots, \phi(c_{i5})^T)^T \in R^d, \quad (2.5)$$

де *d* – загальна розмірність обробленого простору ознак.

Усі ці перетворення закладають математичний фундамент для подальшого побудування класифікаційної моделі, яка навчається на наборі *D* з таких векторів, і дозволяє з високою ймовірністю відрізняти звичайну активність від аномальної, спираючись на статистичні закономірності у вхідних параметрах мережевого трафіку.

Після формування обробленого вектора ознак $x_i \in R^d$, наступним ключовим кроком є побудова класифікатора, який приймає рішення щодо приналежності трафіку до нормальної або аномальної поведінки. У даній роботі використовується модель градієнтного узагальнення рішень у вигляді дерев (FastTree), яка є ітераційною надбудовою над простими деревами рішень.

Алгоритм FastTree формує послідовність дерев $h_1, h_2, \dots, h_M$, кожне з яких прагне уточнити прогноз попередніх. Таким чином, загальна модель виражається як сума виходів усіх дерев:

$$F_M(x) = \sum_{m=1}^{M} \gamma_m \cdot h_m(x), \quad (2.6)$$

де:

M – кількість ітерацій (або дерев);

$h_m(x)$ – прогноз дерева на mm-ій ітерації;

γ_m – вага дерева, що враховує його значущість у загальному рішенні.

Ця формула описує сукупне значення, або «логіт», яке ще не є ймовірністю, але слугує проміжним підрахунком.

Оскільки основна задача – бінарна класифікація, результат у вигляді числа необхідно інтерпретувати як ймовірність того, що трафік є аномальним. Для цього використовується стандартна функція логістичної активації:

$$p(x) = \frac{1}{1+\exp(-F_M(x))}, \quad (2.7)$$

де $p(x) \in (0,1)$ – оціночна ймовірність того, що вектор xx належить до класу аномалії. Таким чином, значення F_M масштабуються до інтервалу від 0 до 1, що дозволяє порівнювати їх зі заздалегідь встановленим порогом.

На основі отриманої ймовірності здійснюється прийняття остаточного рішення щодо класифікації:

$$\hat{y}(x) = \begin{cases} 1, \text{якщо } p(x) \geq \tau, \\ 0, \text{в іншому випадку}, \end{cases} \quad (2.8)$$

де:

$\tau \in (0,1)$ – це порогове значення, яке визначає, наскільки високою має бути ймовірність, щоб зробити висновок про аномалію.

Вибір τ є критичним: низький поріг може призвести до великої кількості хибних спрацювань (false positives), тоді як надто високий – до пропуску реальних загроз (false negatives).

Навчання моделі ґрунтується на мінімізації сукупної похибки, яка вимірюється через крос-ентропійну функцію втрат:

$$L = \sum_{i=1}^{N} [y_i \cdot \ln p(x_i) + (1 - y_i) \cdot \ln(1 - p(x_i))], \quad (2.9)$$

де:

y_i – справжня мітка класу (0 або 1);

$p(x_i)$ – ймовірність, яку передбачає модель.

Ця функція штрафує великі розбіжності між фактичними мітками та ймовірностями, які видає модель. Найменше значення функції досягається тоді, коли модель робить передбачення, максимально наближені до істинних міток.

На кожній ітерації алгоритму виконується побудова дерева, яке апроксимує напрямок покращення (градієнт) функції втрат. Цей напрямок визначається як різниця між фактичною міткою та передбаченою ймовірністю:

$$g_{im} = y_i - p_i^{(m-1)}, \quad (2.9)$$

де:

$p_i^{(m-1)}$ – значення ймовірності на попередньому кроці m−1m−1;

g_{im} – наближене значення похідної функції втрат по моделі, яке відображає, наскільки потрібно «зрушити» прогноз, щоб покращити відповідність істинному значенню.

Саме ці значення $\{g_{im}\}$ використовуються для побудови нового дерева h_m, яке навчається прогнозувати ці «похибки».

Після побудови та навчання моделі постає важливе питання її оцінювання – наскільки точно вона виконує класифікацію нових, невідомих системі

прикладів. Для цього використовується низка формально визначених метричних показників, які дозволяють не лише кількісно порівнювати моделі, а й зважувати їх поведінку в умовах різного дисбалансу між класами. У цьому підрозділі розглянемо наступні п’ять формул, що описують ключові показники якості моделі.

Першим кроком у кількісному аналізі результатів класифікації є визначення кількостей випадків для кожної можливої комбінації істинної мітки y_i та передбаченого значення $\hat{y}_i$ З цією метою вводяться чотири числові характеристики:

– *TP* – кількість прикладів, для яких $y_i = 1$ та $\hat{y}_i = 1$ (модель правильно виявила аномалію);

– *FN* – кількість прикладів, для яких $y_i = 1$ та $\hat{y}_i = 0$ (модель не виявила існуючу аномалію);

– *FP* – кількість прикладів, для яких $y_i = 0$ та $\hat{y}_i = 1$ 1 (модель помилково позначила нормальну активність як аномалію);

– *TN* – кількість прикладів, для яких $y_i = 0$ та $\hat{y}_i = 0$ (модель правильно класифікувала нормальну активність).

Ці значення формують основу для обчислення всіх основних метрик, які кількісно характеризують здатність моделі до правильного розпізнавання аномальних подій у мережевому трафіку.

Для формалізації значень в матриці використовуються наступні обчислення:

$$TP = \sum_{i=1}^{N} I(y_i = 1 \wedge \hat{y}_i = 1), \quad (2.10)$$

$$FP = \sum_{i=1}^{N} I(y_i = 0 \wedge \hat{y}_i = 1), \quad (2.11)$$

$$FN = \sum_{i=1}^{N} I(y_i = 1 \wedge \hat{y}_i = 0), \quad (2.12)$$

$$TN = \sum_{i=1}^{N} I(y_i = 0 \wedge \hat{y}_i = 0, \quad (2.13)$$

де $I(\cdot)$ – індикаторна функція, яка дорівнює 1, якщо умова виконується, і 0 – інакше.

Ці формули дозволяють підрахувати кількість випадків кожного типу класифікації для всієї тестової вибірки. Значення y_i – справжня мітка, а $\hat{y}_i$ – передбачене моделлю значення.

Один з найпростіших, але найпоширеніших показників – точність (accuracy), яка показує частку правильно класифікованих прикладів серед усіх:

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN}. \quad (2.14)$$

Цей показник інформативний тоді, коли класи збалансовані, однак у випадку з виявленням аномалій, де більшість трафіку – нормальний, він може бути вводячим в оману, оскільки модель може мати високу точність при повному ігноруванні аномалій.

Наступним важливим показником є прецизійність, яка визначає, яка частка передбачених аномалій справді є аномаліями:

$$Precision = \frac{TP}{TP+FP}. \quad (2.15)$$

Це співвідношення показує, наскільки модель обережна при винесенні рішень про аномальність. Високе значення прецизійності означає, що серед позначених аномалій мало хибних спрацювань.

Показник повноти відображає, яку частку справжніх аномалій модель змогла виявити:

$$Recall = \frac{TP}{TP+FN}. \quad (2.16)$$

Цей показник важливий з точки зору непропуску загроз: високе значення означає, що модель знаходить більшість справжніх інцидентів. Недоліком високої повноти без супутньої прецизійності може бути надмірна кількість хибних спрацювань.

У сукупності ці показники дозволяють оцінити баланс між здатністю моделі виявляти аномалії та її обережністю при винесенні таких рішень. Найкращі моделі забезпечують компроміс між прецизійністю та повнотою.

Показники прецизійності та повноти характеризують два різні аспекти якості моделі. Проте у практичному застосуванні часто важливо збалансовано враховувати обидва показники одночасно. Для цього використовується F-міра

першого порядку, або просто F_1-міра, яка визначається як гармонічне середнє між прецизійністю та повнотою:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}. \quad (2.16)$$

Формула має наступну властивість: якщо хоча б один з показників є дуже низьким, значення F_1 також буде низьким. Таким чином, F-міра штрафує моделі, які мають сильний перекіс – наприклад, дуже високу прецизійність, але слабку повноту (або навпаки). Це робить її особливо корисною в задачах виявлення аномалій, де бажано забезпечити одночасно і мінімум хибних спрацювань, і мінімум пропущених загроз.

Також важливим засобом оцінювання моделі є побудова кривої "істинно позитивних результатів" залежно від «хибно позитивних». Така крива отримала назву ROC-кривої (від англ. Receiver Operating Characteristic), і вона будується на основі порівняння наступних величин для різних значень порогу τ:

$$TPR = \frac{TP}{FP + TN}. \quad (2.17)$$

Частка хибних позитивних рішень серед усіх нормальних прикладів:

$$FPR = \frac{FP}{FP + TN}. \quad (2.18)$$

Змінюючи поріг τ від 0 до 1, для кожного його значення обчислюється пара (FPR,TPR), що дозволяє побудувати відповідну криву. Вона демонструє, як змінюється чутливість моделі за умови зростання її схильності до хибних спрацювань.

Одним з найстабільніших та найінформативніших показників якості класифікатора є площа під ROC-кривою, яка формально визначається як:

$$AUC = \int_0^1 TPR(FPR) d(FPR).. \quad (2.19)$$

Ця величина AUC∈[0,1]AUC∈[0,1] оцінює здатність моделі розділяти класи незалежно від вибраного порогу. У практиці:

AUC≈0.5 – модель не краща за випадкове вгадування;

AUC≈1.0 – майже ідеальне розділення класів;

AUC<0.5 – модель має зворотну інтерпретацію (класи переплутано).

Описана модель дозволяє виявляти аномальну активність у мережі, спираючись на ключові характеристики трафіку та їхню статистичну поведінку. Вона навчається на прикладах і оцінює ймовірність загрози, а її якість перевіряється за показниками точності, чутливості та здатності відрізняти норму від відхилення. Такий підхід забезпечує надійне виявлення потенційних загроз у реальному часі.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розділ присвячено безпосередній реалізації програмного забезпечення, що виконує функції виявлення аномалій у мережевому трафіку на основі алгоритмів машинного навчання. На основі сформованих вимог, спроєктованої архітектури та обґрунтованого вибору технологій реалізовано модулі збору, обробки, класифікації та виведення даних, що інтегруються в єдину систему з графічним інтерфейсом користувача. Особливу увагу приділено реалізації навчального конвеєра, взаємодії з моделлю під час моніторингу та збереженню результатів у базу даних.

3.1 Реалізація модуля обробки вхідних логів і дампів трафіку

У межах побудови системи виявлення аномалій ключовим компонентом є модуль обробки вхідних логів і дампів мережевого трафіку, який забезпечує аналітичну підготовку інформації до подальшого машинного аналізу. Цей модуль відповідає за початкову інтерпретацію структурованих та напівструктурованих даних, їх фільтрацію, попередню класифікацію, а також за формування агрегованих оцінок, що використовуються як у виведенні результатів, так і для формування вхідних векторів у моделей прогнозування. Надійна реалізація цього блоку дозволяє здійснювати первинне оцінювання без залучення повноцінного ML-конвеєра, що підвищує загальну стійкість та автономність системи. Архітектурно він є частиною бізнес-логіки і передбачає комбінування емпіричних правил із формалізованими структурними типами.

На рис. 3.1 наведено ключовий фрагмент коду, що визначає основні структури для оцінки ризику та навантаження на мережу.

```
public enum TrafficLoadLevel { Low, Moderate, High }

/// <summary>Ключові параметри оцінювання ризику.</summary>
3 references
internal struct RiskAssessment {
  public double ErrorRatePercent;   // % з'єднань, що завершились помилкою
  public double LoadScore;          // 0–1, нормалізований рівень навантаження
  public int CompositeScore;     // 0–100, інтегральний бал
  public string Recommendation;     // базова порада
}
```

Рисунок 3.1 – Фрагмент коду оцінювання мережевого ризику

Перелік можливих рівнів мережевого навантаження у вигляді перерахування TrafficLoadLevel включає три категорії: низьке, помірне та високе навантаження. Ці значення можуть використовуватись як у простій класифікації, так і як ознака для алгоритмів прийняття рішень. Для комплексного опису ризикового профілю передбачено структуру RiskAssessment, яка містить чотири основні параметри: відсоток помилкових з'єднань (ErrorRatePercent), нормалізовану оцінку навантаження (LoadScore), інтегральний ризиковий бал (CompositeScore) та текстову рекомендацію щодо подальших дій (Recommendation). Завдяки структурованій формі представлення ці дані можуть легко передаватися між компонентами системи, зберігатися в логах або відображатися в інтерфейсі користувача для аналітичного реагування. Така реалізація забезпечує гнучкість, розширюваність і спрощує подальшу інтеграцію з ML-моделлю.

На рис. 3.2 відображено метод, який реалізує механізм класифікації рівня навантаження на основі загального обсягу мережевого трафіку, вираженого в мегабайтах.

```
private TrafficLoadLevel ClassifyLoad(long totalBytes) {
  double mb = totalBytes / (double)MB;
  if (mb < LOAD_LOW_MB) return TrafficLoadLevel.Low;
  if (mb < LOAD_HIGH_MB) return TrafficLoadLevel.Moderate;
  else return TrafficLoadLevel.High;
}
```

Рисунок 3.2 – Класифікація рівня навантаження за обсягом трафіку

Метод приймає на вхід значення загальної кількості байтів, після чого виконує його перетворення у мегабайти шляхом поділу на фіксовану константу. Отримане значення порівнюється з порогами, визначеними для умовно низького та високого навантаження. У разі, якщо обсяг менший за нижню межу, трафік класифікується як низький; якщо ж він перевищує нижню, але не перевищує верхню межу – як помірний; і, відповідно, якщо перевищено верхній поріг – як високий.

На рис. 3.3 наведено метод оцінювання ризику, який виконує комбіновану інтерпретацію мережевого навантаження та частки помилкових з'єднань для формування інтегральної оцінки потенційної загрози. Метод

приймає як вхідні дані загальну статистику трафіку (TrafficStats) та попередньо визначений рівень навантаження (TrafficLoadLevel), а у відповідь повертає структуру RiskAssessment, що містить усі ключові індикатори ризику.

```
private RiskAssessment AssessRisk(TrafficStats s, TrafficLoadLevel load) {
  var ra = new RiskAssessment();
  // 1) Error Rate, %
  ra.ErrorRatePercent =
    s.TotalConnections == 0 ? 0 :
    (double)s.ErrorCount / s.TotalConnections * 100.0;
  // 2) Load Score, 0–1
  double mb = s.TotalBytes / (double)MB;
  double loadScore =
        mb <= LOAD_LOW_MB ? 0.2
      : mb >= LOAD_HIGH_MB ? 1.0
      : (mb - LOAD_LOW_MB) / (LOAD_HIGH_MB - LOAD_LOW_MB); // лінійно
  ra.LoadScore = loadScore;
  // 3) Composite Score: 70% – навантаження, 30% – помилки
  ra.CompositeScore = (int)(loadScore * 70
                      + Math.Min(ra.ErrorRatePercent, 100) * 0.30);
  // 4) Текстова рекомендація
  ra.Recommendation =
    ra.CompositeScore >= 70 ? "Рекомендується негайний аналіз трафіку."
  : ra.CompositeScore >= 40 ? "Слід перевірити конфігурацію та логи."
                            : "Ситуація в межах норми.";
  return ra;
}
```

Рисунок 3.3 – Код методу оцінювання ризику

На першому етапі розраховується відсоток помилок, як частка помилкових з’єднань від загальної кількості, з перетворенням у відсоткове значення. Далі визначається нормалізована оцінка навантаження: якщо трафік нижчий за нижній поріг, коефіцієнт навантаження дорівнює 0.2; якщо перевищує верхній – приймає значення 1.0; у проміжку між порогами – обчислюється за лінійною інтерполяцією. Отримані значення використовуються для формування інтегрального ризикового балу, де 70 % ваги надається навантаженню, а 30 % – частці помилок. Після цього генерується текстова рекомендація: за високого балу (70 і більше) система пропонує негайний аналіз трафіку; при помірному ризику (від 40 до 69) – перевірку конфігурації; у випадку низького ризику – фіксується нормальний стан. Метод реалізує просту, проте достатньо показову модель оцінювання загального ризикового профілю мережевої активності.

На рис. 3.4 подано метод формування аналітичного звіту за IP-адресою, який є публічним інтерфейсом модуля оцінювання мережевої активності. Метод приймає IP-адресу як вхідний параметр, ініціює отримання статистики

трафіку через об’єкт TrafficStats, класифікує рівень навантаження за обсягом переданих даних, викликає механізм оцінки ризику, а далі формує текстовий звіт з детальними характеристиками та висновками щодо поточного стану.

```
public string GenerateReportByIP(string ip) {
  TrafficStats s = TrafficStatsProvider.GetStats(ip);
  // --- Аналітична частина ---
  TrafficLoadLevel load = ClassifyLoad(s.TotalBytes);
  RiskAssessment ra = AssessRisk(s, load);
  // --- Формування звіту ---
  var sb = new StringBuilder()
    .AppendLine($"IP-адреса: {ip}")
    .AppendLine($"Час останнього запиту: {s.LastRequest:dd.MM.yyyy HH:mm:ss}")
    .AppendLine()
    .AppendLine("*** Показники з'єднань ***")
    .AppendLine($"— Загальна кількість з'єднань:        {s.TotalConnections}")
    .AppendLine($"   • Вхідні:                          {s.InboundConnections}")
    .AppendLine($"   • Вихідні:                         {s.OutboundConnections}")
    .AppendLine($"— Кількість унікальних портів:        {s.UniquePorts}")
    .AppendLine($"— Помилки / дропи:                    {s.ErrorCount} " +
                $"({ra.ErrorRatePercent:F2} %)")
    .AppendLine()
    .AppendLine("*** Обсяг трафіку ***")
    .AppendLine($"— Загалом передано даних:             {s.TotalBytes:N0} байт")
    .AppendLine($"   • Вхідний (IN):                    {s.InboundBytes:N0} байт")
    .AppendLine($"   • Вихідний (OUT):                  {s.OutboundBytes:N0} байт")
    .AppendLine($"— Пікова пропускна здатність:         {s.PeakThroughput:N0} байт/хв")
    .AppendLine($"— Середній розмір пакета:             {s.AvgPacketSize:N0} байт")
    .AppendLine($"— Рівень навантаження:                {load}")
    .AppendLine()
    .AppendLine("*** Оцінка ризику ***")
    .AppendLine($"— Композитний ризиковий бал:          {ra.CompositeScore}/100")
    .AppendLine($"— Рекомендація:                       {ra.Recommendation}");

  return sb.ToString();
}
```

Рисунок 3.4 – Формування аналітичного звіту за IP-адресою

У структурі звіту відображається дата та час останнього запиту, загальна кількість з’єднань із розподілом на вхідні та вихідні, кількість унікальних портів і зафіксованих помилок, а також обсяг переданого трафіку у байтах, включно з піковим навантаженням і середнім розміром пакета. Окремо подається рівень навантаження у якісному вигляді (низьке, помірне, високе), обчислений композитний ризиковий бал у шкалі від 0 до 100, а також текстова рекомендація для адміністратора. Метод дозволяє в інтегрованій формі поєднати кількісні та якісні показники мережевої активності, що робить його придатним для оперативного аналізу ситуації без необхідності візуалізації чи машинного розбору.

3.2 Реалізація модуля машинного навчання для виявлення аномалій

У межах побудови програмного забезпечення для виявлення аномалій у мережевому трафіку центральне місце займає модуль машинного навчання, який забезпечує навчання, збереження та подальше використання моделі класифікації. Його реалізація передбачає інтеграцію функціоналу бібліотеки ML.NET у застосунок із графічним інтерфейсом, що дозволяє користувачеві працювати з навчальними вибірками, запускати процес тренування моделі, зберігати результат і використовувати її для прогнозування безпосередньо з форми. На цьому рівні також реалізуються засоби обробки CSV-файлів, формування ознак, інженерії даних і перевірки структури набору.

Першим кроком у взаємодії користувача з модулем є завантаження навчального набору даних, інтерфейс якого реалізовано у вигляді кнопки з обробником події OpenBtn_Click, наведеної на рис. 3.5.

```
private void OpenBtn_Click(object sender, EventArgs e) {
  // Створення діалогового вікна для відкриття файлу
  OpenFileDialog openFileDialog = new OpenFileDialog {
    Filter = "Text files (*.csv)|*.csv|All files (*.*)|*.*",
    FilterIndex = 2,
    RestoreDirectory = true
  };
  if (openFileDialog.ShowDialog() == DialogResult.OK) {
    _Path = openFileDialog.FileName;
    FileNameTBox.Text = openFileDialog.FileName;
```

Рисунок 3.5 – Фрагмент коду завантаження навчального набору

Даний метод відкриває стандартне діалогове вікно вибору файлу, фільтруючи дозволені типи розширень до CSV-файлів, хоча користувач також може обрати інший тип через опцію «усі файли». Після підтвердження вибору система зчитує повний шлях до обраного файлу та зберігає його у змінну _Path для подальшого використання під час тренування. Назва файлу також відображається у відповідному текстовому полі інтерфейсу, що дозволяє користувачеві впевнитися у коректності вибору. Цей фрагмент є початковою точкою навчального процесу, який забезпечує інтерактивність і підвищує зручність експлуатації системи.

Рис. 3.6 відображає фрагмент коду, який демонструє початкову ініціалізацію процесу машинного навчання після вибору файлу з даними. Цей

код відповідає за створення контексту ML.NET і завантаження вхідного набору для подальшого аналізу та тренування моделі.

```
mlContext = new MLContext(seed: 1);
// Завантаження даних
dataView = mlContext.Data.LoadFromTextFile<NetworkTrafficData>(
    path: _Path,
    hasHeader: true,
    separatorChar: ',');
RaportTBox.Text = "Завантаження даних\r\n";
Application.DoEvents();
```

Рисунок 3.6 – Ініціалізацію процесу навчання та завантаження даних

У першому рядку створюється об’єкт MLContext із фіксованим параметром генератора випадкових чисел, що забезпечує повторюваність результатів під час навчання. Далі, за допомогою методу LoadFromTextFile<T>, виконується завантаження CSV-файлу, де вказано наявність заголовку та роздільник. Тип даних, з якими працює модель, визначається через шаблон NetworkTrafficData, який містить відповідні поля трафіку. Після успішного завантаження вмісту до структури IDataView, в текстове поле звіту виводиться підтвердження про завершення цього етапу, а виклик Application.DoEvents() забезпечує негайне оновлення інтерфейсу користувача без блокування подальших дій.

На рис. 3.7 наведено фрагмент коду, що передує побудові навчального конвеєра та виконує попередній аналіз даних із навчальної вибірки. Цей етап дозволяє визначити, які ознаки будуть використовуватися як числові в процесі тренування моделі, а також вивести статистику розподілу прикладів за класами, що є важливим для розуміння збалансованості датасету.

```
string[] numericColumns = {
            "avg_ipt", "bytes_in", "bytes_out", "entropy",
            "num_pkts_out", "num_pkts_in", "total_entropy", "duration"
        };
// Виведення кількості взірців для кожного класу
var labelCounts =
  mlContext.Data.CreateEnumerable<NetworkTrafficData>(dataView, reuseRowObject: false)
    .GroupBy(data => data.label)
    .Select(group => new { Label = group.Key, Count = group.Count() });
RaportTBox.Text += "Кількість взірців:\r\n";
foreach (var count in labelCounts) {
  RaportTBox.Text += ($"Клас: {count.Label}, Кількість: {count.Count}\r\n");
}
```

Рисунок 3.7 – Категоризація атрибутів та виведення інформації про взірці

У масиві numericColumns задається перелік ознак, які є числовими та будуть використовуватись для побудови векторів ознак у моделі машинного навчання. Далі створюється колекція типу IEnumerable<NetworkTrafficData>, що дозволяє ітеруватися по всіх прикладах з набору dataView, після чого вони групуються за міткою класу label. Для кожної групи підраховується кількість елементів, і результати виводяться до текстового поля звіту. Такий підхід дозволяє отримати уявлення про наявність дисбалансу класів, що може вплинути на якість класифікації під час навчання моделі.

Рис. 3.8 відображає побудову конвеєра обробки даних, що є необхідним етапом перед навчанням моделі машинного навчання. Цей конвеєр формує єдиний потік перетворень, які застосовуються до початкового датасету для формування коректних вхідних ознак у форматі, що відповідає вимогам алгоритму класифікації. Його основна мета – уніфікація вхідних ознак, кодування категоріальних значень, конкатенація числових полів та нормалізація.

```
var dataProcessPipeline = mlContext.Transforms.Conversion.MapValue(
        outputColumnName: "Label",
        inputColumnName: "label",
        keyValuePairs: new[] {
                    new KeyValuePair<string, bool>("benign", false),
                    new KeyValuePair<string, bool>("outlier", true)
        })
    .Append(mlContext.Transforms.Categorical.OneHotEncoding("ProtoEncoded", "proto"))
    .Append(mlContext.Transforms.Categorical.OneHotHashEncoding("DestIpEncoded",
    "dest_ip", numberOfBits: 10))
    .Append(mlContext.Transforms.Concatenate("Features", numericColumns))
    .Append(mlContext.Transforms.NormalizeMeanVariance("Features"));
```

Рисунок 3.8 – Побудова конвеєра обробки даних

У першій частині конвеєра відбувається перетворення цільової змінної label у булевий формат Label, де «benign» зіставляється зі значенням false, а «outlier» – з true. Далі виконується перетворення протоколу (proto) за допомогою one-hot кодування, що дозволяє представити його як набір бінарних ознак. Наступним кроком є хеш-кодування IP-адреси призначення (dest_ip) у вектор фіксованої розмірності (10 біт), що зменшує розмірність у порівнянні з класичним one-hot. Потім усі числові ознаки об’єднуються в єдиний вектор Features, після чого до нього застосовується нормалізація за середнім і

дисперсією, що дозволяє уникнути домінування ознак з великими масштабами під час навчання моделі. Цей етап є критично важливим для побудови стабільного та узагальнюючого алгоритму класифікації.

На рис. 3.9 зображено завершальний етап побудови повного навчального конвеєра, що включає вибір алгоритму класифікації, об'єднання його з конвеєром обробки даних, а також розділення датасету на навчальну та тестову вибірки. Це є підготовчим кроком до безпосереднього тренування моделі, який забезпечує відокремлення етапів перетворення ознак та навчання.

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

var trainingPipeline = dataProcessPipeline.Append(trainer);
// Розділення даних
var split = mlContext.Data.TrainTestSplit(dataView, testFraction: 0.2);
var trainData = split.TrainSet;
var testData = split.TestSet;
```

Рисунок 3.9 – Вибір алгоритму класифікації та розділення даних

У цьому фрагменті обрано бінарний класифікатор FastTree, що реалізує метод градієнтного бустингу на базі дерев рішень і працює з ознаками у векторі Features та мітками в полі Label. Конвеєр навчання (trainingPipeline) формується шляхом послідовного з'єднання обробника даних (dataProcessPipeline) з обраним тренером. Після цього датасет dataView розділяється на навчальну та тестову частини у співвідношенні 80% до 20%, що дозволяє забезпечити незалежну оцінку якості моделі на відкладених даних. Отримані підмножини trainData та testData використовуються в наступних етапах – тренування моделі, оцінювання її продуктивності та збереження для подальшого використання.

Рис. 3.10 відображає фрагмент коду, який реалізує завершальний етап побудови моделі машинного навчання – процес її навчання та подальше оцінювання якості на тестовій вибірці.

```
trainedModel = trainingPipeline.Fit(trainData);

// Оцінювання моделі
var predictions = trainedModel.Transform(testData);
var metrics =
  mlContext.BinaryClassification.Evaluate(predictions, labelColumnName: "Label");
```

Рисунок 3.10 – Навчання та оцінювання моделі

У першому рядку здійснюється навчання моделі шляхом виклику методу Fit на побудованому конвеєрі trainingPipeline, використовуючи попередньо сформовану навчальну вибірку trainData. Результатом є об’єкт trainedModel, який містить усі обробники, кодувальники та навчений класифікатор. Після цього на тестових даних testData виконується операція Transform, яка застосовує модель для передбачення класів, а результат зберігається в об’єкті predictions. Метод Evaluate використовується для обчислення метрик якості класифікації, таких як точність, повнота, F-міра, AUC та інші, що дозволяє кількісно оцінити здатність моделі узагальнювати закономірності з навчальних даних.

На рис. 3.11 зображено фрагмент коду, який реалізує виведення аналітичних результатів оцінювання моделі після її тренування. Цей блок відповідає за формування звітної частини, яка включає як загальні метрики класифікації, так і деталізовану інформацію щодо помилок класифікації та характеристик для кожного класу. Всі результати відображаються у текстовому полі RaportTBox у зручному для користувача форматі.

```
RaportTBox.Text += "=== Загальні метрики ===\r\n";
RaportTBox.Text += ($"Accuracy: {metrics.Accuracy:P2}\r\n");
RaportTBox.Text += ($"AUC: {metrics.AreaUnderRocCurve:P2}\r\n");
RaportTBox.Text += ($"F1 Score: {metrics.F1Score:P2}\r\n");
// Виведення матриці помилок
RaportTBox.Text += "=== Матриця помилок ===\r\n";
var scoredData =
  mlContext.Data.CreateEnumerable<NetworkTrafficPrediction>(predictions,
  reuseRowObject: false).ToList();
var truePositives = scoredData.Count(p => p.PredictedLabel && p.Probability >= 0.5);
var trueNegatives = scoredData.Count(p => !p.PredictedLabel && p.Probability < 0.5);
var falsePositives = scoredData.Count(p => p.PredictedLabel && p.Probability < 0.5);
var falseNegatives = scoredData.Count(p => !p.PredictedLabel && p.Probability >= 0.5);
RaportTBox.Text += ($"True Positives: {truePositives}\r\n");
RaportTBox.Text += ($"True Negatives: {trueNegatives}\r\n");
RaportTBox.Text += ($"False Positives: {falsePositives}\r\n");
RaportTBox.Text += ($"False Negatives: {falseNegatives}\r\n");
// Виведення метрик для кожного класу
RaportTBox.Text += "=== Метрики для кожного класу ===\r\n";
RaportTBox.Text += "Позитивний клас (1):\r\n";
RaportTBox.Text += ($"Precision: {metrics.PositivePrecision:P2}\r\n");
RaportTBox.Text += ($"Recall: {metrics.PositiveRecall:P2}\r\n");
RaportTBox.Text += "Негативний клас (0):\r\n";
RaportTBox.Text += ($"Precision: {metrics.NegativePrecision:P2}\r\n");
RaportTBox.Text += ($"Recall: {metrics.NegativeRecall:P2}\r\n");
```

Рисунок 3.11 – Виведення аналітичних результатів оцінювання моделі

У першій частині звіту виводяться загальні метрики: точність (Accuracy), площа під кривою ROC (AUC) та середньозважена F-міра (F1 Score), що дають

узагальнену оцінку здатності моделі до класифікації. Далі обчислюється матриця помилок, яка базується на значеннях передбаченої мітки (PredictedLabel) та порогового значення ймовірності (Probability). На основі цього визначаються кількості істинно позитивних, істинно негативних, хибно позитивних та хибно негативних передбачень. Після цього звіт доповнюється метриками точності та повноти окремо для кожного класу (0 – нормальна активність, 1 – аномалія), що дозволяє детально оцінити якість класифікації в умовах дисбалансу класів. Такий формат представлення результатів є цінним для подальшого аналізу ефективності побудованої моделі.

На рис. 3.12 подано метод, який реалізує завершальний етап взаємодії з модулем машинного навчання – збереження навченої моделі та відповідних метаданих. Метод прив’язано до події натискання кнопки збереження і включає як перевірку коректності введених даних, так і серіалізацію моделі у файл для подальшого використання. Він також забезпечує інтеграцію з базою даних для фіксації події навчання, що є важливим для ведення історії змін та аудиту дій користувача.

```
private void SaveBtn_Click(object sender, EventArgs e) {
  if (IsDataEnteringCorrect()) {
    //Зберігання моделі
    string pathName = @"\teach\" + GenerateFileName() + ".zip";
    string localProj =
      System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location);
    _ModelsProvider.InsertModels(ModelsNamesTBox.Text, pathName);
    mlContext.Model.Save(trainedModel, dataView.Schema, localProj + pathName);
    ClearAllData();
    _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
      "Було навчено модель " +
      ModelsNamesTBox.Text, DateTime.Now);
    MessageBox.Show("Дані успішно збережено!");
  }
}
```

Рисунок 3.12 – Зберігання моделі

У випадку, якщо вхідні дані проходять перевірку, генерується унікальна назва файлу для моделі, після чого формується абсолютний шлях для збереження у спеціальній директорії проєкту. Назва моделі та шлях до неї додаються в базу даних за допомогою методу InsertModels, після чого модель серіалізується методом mlContext.Model.Save. Очищення полів введення забезпечує готовність інтерфейсу до наступної сесії, а інформація про

створення моделі записується у журнал подій за допомогою InsertLogs. Після завершення усіх операцій користувач отримує підтвердження про успішне збереження у вигляді повідомлення. Цей метод є завершальним кроком у циклі навчання та дозволяє підтримувати колекцію моделей для подальшого прогнозування.

Рис. 3.13 висвітлює метод, що відповідає за завантаження раніше збереженої моделі машинного навчання та підготовку її до використання для прогнозування в режимі реального часу. Метод викликається при виборі моделі для моніторингу або при повторному запуску застосунку, що дозволяє миттєво активувати механізм інференсу без необхідності повторного навчання.

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

Рисунок 3.13 – Завантаження збереженої моделі машинного навчання

На початку формується абсолютний шлях до збереженого файлу моделі, який міститься у директорії запуску застосунку. Далі виконується завантаження моделі за допомогою mlContext.Model.Load, результатом чого є об’єкт ITransformer, а також схема даних (DataViewSchema), необхідна для сумісності з форматом ознак. Після успішного завантаження створюється PredictionEngine – механізм, який приймає об’єкти типу NetworkTrafficData та повертає результат передбачення у вигляді NetworkTrafficPrediction. Цей метод ініціалізує прогнозувальну підсистему, забезпечуючи готовність системи до аналізу мережевої активності.

На рис. 3.14 зображено метод, що реалізує керування запуском та зупинкою моніторингу мережевої активності у вигляді симульованого надходження тестових даних. Цей метод прив’язаний до кнопки MonitoringBtn у графічному інтерфейсі та дозволяє перемикати стан таймера, який відповідає

за періодичне генерування вхідних даних для моделі. Основна мета цього механізму – забезпечити інтерактивне тестування працездатності моделі в умовах, близьких до реального часу, без підключення до зовнішніх джерел трафіку.

```
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);
    }
  }
}
```

Рисунок 3.14 – Керування запуском та зупинкою моніторингу мережі

У разі, якщо обрана модель існує, перевіряється поточний стан таймера MoniroringTimer. Якщо він активний, то його виконання зупиняється, кнопка змінює свій напис на «Моніторити», а у журнал подій записується відповідна інформація про припинення генерації. Якщо ж таймер вимкнений, то навпаки – активується симуляція надходження даних, інтерфейс оновлюється, і подія запуску фіксується у логах із прив’язкою до імені поточної моделі. Такий підхід дозволяє гнучко керувати процесом тестування та аналізувати реакцію системи на умовні аномалії у трафіку.

На рис. 3.15 подано фрагмент коду, що демонструє формування вхідного об’єкта NetworkTrafficData, який буде передано у модель машинного навчання для прогнозування. Цей об’єкт створюється шляхом копіювання значень із іншого джерела даних – змінної data, яка містить симульовану або реальну інформацію про мережеву активність. Такий підхід дозволяє уніфікувати структуру даних, забезпечивши повну відповідність між форматом, що очікується моделлю, і фактичними вхідними значеннями.

```
NetworkTrafficData mlData = new NetworkTrafficData {
  avg_ipt = data.avg_ipt,
  bytes_in = data.bytes_in,
  bytes_out = data.bytes_out,
  dest_ip = data.dest_ip,
  dest_port = data.dest_port,
  entropy = data.entropy,
  num_pkts_out = data.num_pkts_out,
  num_pkts_in = data.num_pkts_in,
  proto = data.proto,
  src_ip = data.src_ip,
  src_port = data.src_port,
  label = data.label,
  total_entropy = data.total_entropy,
  duration = data.duration
};
```

Рисунок 3.15 – Формування вхідного об’єкта NetworkTrafficData

Усі властивості – як числові (наприклад, avg_ipt, bytes_in, entropy), так і категоріальні (proto, src_ip) – явно присвоюються новоствореному об’єкту, що дозволяє уникнути автоматичних або неявних перетворень, які можуть призвести до втрати точності або несумісності. Таке пряме клонування значень особливо важливе при моделюванні передбачень у реальному часі або в межах системи, що працює з декількома джерелами даних одночасно. Об’єкт mlData, сформований у такий спосіб, є повністю готовим до передачі в механізм PredictionEngine.

Рис. 3.16 відображає фрагмент коду, який реалізує процес інференсу моделі машинного навчання над попередньо сформованими вхідними даними. У цьому коді використовується об’єкт predictionEngine, що створений раніше, для обчислення передбачення на основі об’єкта mlData. Для вимірювання продуктивності виконується запуск таймера Stopwatch, що дозволяє визначити час, витрачений на прогнозування.

```
  MonitoringTBox.Text = dataInfo.ToString();
  var stopwatch = System.Diagnostics.Stopwatch.StartNew();
  var prediction = predictionEngine.Predict(mlData);
  stopwatch.Stop();
  double elapsedMs = stopwatch.Elapsed.TotalMilliseconds;
  var answer = new StringBuilder();
  answer.AppendLine("\r\n--- Прогнозування ---");
  answer.AppendLine($"\r\nЄ аномалія?: {(prediction.PredictedLabel ? "Так" : "Ні")}");
  answer.AppendLine($"\r\nЙмовірність аномалії: {prediction.Probability:P2}");
  answer.AppendLine($"\r\nЧас прогнозування: {elapsedMs:F3} мс"); // 3 знаки після коми
  // Виведення результату прогнозування
  MonitoringTBox.Text += answer.ToString();
  if (prediction.PredictedLabel) {
    AlertIntrusion(mlData, prediction.Probability);
  }
}));
```

Рисунок 3.16 – Процес прогнозування анамалії

Результати передбачення формуються у вигляді текстового повідомлення, яке включає ознаку наявності аномалії (на підставі булевого значення PredictedLabel), ймовірність її виникнення (Probability), а також час обчислення у мілісекундах із точністю до трьох знаків після коми. Згенерований звіт додається до поля моніторингу MonitoringTBox, що відображає історію прогнозів у реальному часі. У випадку, якщо система виявила аномалію, додатково викликається процедура AlertIntrusion, яка може відповідати за логування, сповіщення або візуальну індикацію інциденту. Цей блок завершує повний цикл взаємодії моделі з системою в продуктивному режимі.

3.3 Реалізація модуля забезпечення безпеки

У межах побудови системи виявлення аномалій у мережевому трафіку важливою складовою є реалізація модуля забезпечення безпеки, який відповідає за захист конфіденційних даних користувачів та системних компонентів від несанкціонованого доступу. З огляду на наявність автентифікації, збереження чутливої інформації (зокрема паролів користувачів) та логування ключових дій, виникає потреба в надійному шифруванні. Для реалізації цієї функціональності в системі використовується алгоритм симетричного блочного шифрування AES (Advanced Encryption Standard), що забезпечує високу стійкість до криптоаналізу та підтримується на рівні системних бібліотек .NET.

На рис. 3.17 зображено фрагмент методу Encrypt, який виконує шифрування текстового рядка з використанням криптографічного провайдера AesCryptoServiceProvider. У методі спочатку перевіряється коректність вхідного значення – якщо рядок порожній або має значення null, викликається виняток із поясненням. Далі створюється об'єкт aes, у якому задаються ключ (Key) та ініціалізаційний вектор (IV), які використовуються для забезпечення цілісності та унікальності операцій шифрування. Після цього дані будуть перетворені у байтовий масив, зашифровані в потоковому режимі та повернуті у вигляді закодованого рядка.

```
public string Encrypt(string originalString) {
  // Перевірка, чи вхідний рядок не є null або порожнім
  if (String.IsNullOrEmpty(originalString)) {
    throw new ArgumentNullException("The string which needs to be encrypted can not be null.");
  }
  // Використання AES шифрування з використанням CryptoServiceProvider
  using (AesCryptoServiceProvider aes = new AesCryptoServiceProvider()) {
    // Встановлення ключа та IV для шифрування
    aes.Key = Key;
    aes.IV = IV;
```

Рисунок 3.17 – Фрагмент методу Encrypt

Рис. 3.18 відображає основну частину методу шифрування, що реалізує перетворення відкритого тексту у захищений формат за допомогою симетричного алгоритму AES. У цьому фрагменті використано клас AesCryptoServiceProvider, який надає реалізацію стандарту AES у середовищі .NET. Після ініціалізації об'єкта задаються криптографічний ключ (Key) та вектор ініціалізації (IV), які є критично важливими для забезпечення унікальності та стійкості кожної операції шифрування.

```
using (AesCryptoServiceProvider aes = new AesCryptoServiceProvider()) {
  // Встановлення ключа та IV для шифрування
  aes.Key = Key;
  aes.IV = IV;

  // Створення потоку в пам'яті для зберігання зашифрованих даних
  MemoryStream memoryStream = new MemoryStream();
  // Створення потоку шифрування
  CryptoStream cryptoStream = new CryptoStream(memoryStream,
      aes.CreateEncryptor(aes.Key, aes.IV), CryptoStreamMode.Write);
  // Запис вхідного рядка в потік шифрування
  StreamWriter writer = new StreamWriter(cryptoStream);
  writer.Write(originalString);
  writer.Flush();
  // Завершення шифрування
  cryptoStream.FlushFinalBlock();
  writer.Flush();
  // Повернення зашифрованих даних у форматі Base64
  return Convert.ToBase64String(memoryStream.ToArray());
}
```

Рисунок 3.18 – Основна частина методу шифрування

Шифрування здійснюється в потоковому режимі: спочатку створюється об'єкт MemoryStream, який виступає буфером для зберігання зашифрованих байтів. Далі створюється CryptoStream, який прив’язується до цього буфера та виконує шифрування за допомогою методу CreateEncryptor. Вхідний рядок записується у цей потік через StreamWriter, після чого відбувається примусове очищення буфера та завершення шифрування методом FlushFinalBlock. Отримані байти з пам’яті перетворюються у рядок Base64, що дозволяє зручно

зберігати результат у текстовій формі (наприклад, у базі даних або конфігураційних файлах). Така реалізація забезпечує повноцінний цикл шифрування з дотриманням криптографічних стандартів.

На рис. 3.19 зображено метод Decrypt, який реалізує зворотну операцію до шифрування – розшифрування даних, закодованих у форматі Base64. Метод забезпечує відновлення оригінального тексту, який був попередньо зашифрований із використанням симетричного алгоритму AES. Перед початком розшифрування виконується перевірка вхідного параметра на наявність значення: якщо передано порожній або null-рядок, генерується виняток, що унеможливлює подальше некоректне виконання операції.

```
public string Decrypt(string cryptedString) {
  // Перевірка, чи вхідний рядок для розшифрування не є null або порожнім
  if (String.IsNullOrEmpty(cryptedString)) {
    throw new ArgumentNullException("The " +
      "string which needs to be decrypted can not be null.");
  }
        // Конвертація рядка з Base64 у масив байтів
  var fullCipher = Convert.FromBase64String(cryptedString);
  // Використання AES шифрування з використанням CryptoServiceProvider
  using (AesCryptoServiceProvider aes = new AesCryptoServiceProvider()) {
    // Встановлення ключа та IV для розшифрування
    aes.Key = Key;
    aes.IV = IV;
```

Рисунок 3.19 – Фрагмент методу Decrypt

Рядок, отриманий у вигляді Base64, конвертується у масив байтів, який містить зашифровану інформацію. Створюється екземпляр AesCryptoServiceProvider, для якого встановлюються раніше визначені ключ (Key) та ініціалізаційний вектор (IV). Ці параметри мають точно відповідати тим, що використовувалися при шифруванні, інакше розшифрування буде неможливим або некоректним. Саме цей блок ініціалізації криптографічного об'єкта є підготовчим кроком до основної операції відновлення початкового рядка.

Рис. 3.20 висвітлює завершальну частину методу Decrypt, яка реалізує основну логіку розшифрування зашифрованого тексту з використанням потоку пам’яті та симетричного декодування. Після попередньої ініціалізації параметрів AES створюється об'єкт MemoryStream, до якого передається масив байтів fullCipher, що містить зашифровану інформацію. Цей потік слугує

джерелом для наступного розшифрування, відтворюючи байтовий потік у режимі читання.

```
MemoryStream memoryStream = new MemoryStream(fullCipher);
// Створення потоку розшифрування
CryptoStream cryptoStream = new CryptoStream(memoryStream,
    aes.CreateDecryptor(aes.Key, aes.IV), CryptoStreamMode.Read);
// Читання розшифрованих даних
StreamReader reader = new StreamReader(cryptoStream);
return reader.ReadToEnd(); // Повернення розшифрованого рядка
```

Рисунок 3.20 – Створення потоку та повернення розшифрованого рядка

До створеного потоку пам’яті підключається CryptoStream, який виконує розшифрування даних у режимі Read за допомогою методу CreateDecryptor. Потім створюється об'єкт StreamReader, що дозволяє зчитувати з цього потоку результат у вигляді текстового рядка. Метод повертає повністю відновлений, розшифрований текст, що був збережений у зашифрованій формі. Таким чином, повний цикл відновлення завершено – дані перетворено з захищеного формату назад до читабельного вигляду, придатного для подальшого використання у системі.

4 ТЕСТУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

4.1 Модульне тестування основних компонентів

У процесі розроблення програмного забезпечення для виявлення аномалій у мережевому трафіку особливе значення має перевірка коректності роботи окремих компонентів системи. З цією метою було проведено модульне тестування, яке дозволяє локалізовано перевірити поведінку методів і класів без взаємозв'язку з іншими частинами програми. Такий підхід дає змогу виявити логічні помилки, забезпечити надійність бізнес-логіки та спростити подальшу модифікацію або розширення функціональності. Для реалізації тестування було використано фреймворк MSTest, який інтегрується в середовище Visual Studio та підтримує автоматизоване виконання тестів.

На рис. 4.1 наведено знімок вікна Test Explorer, що демонструє результати запуску 16 модульних тестів, реалізованих у рамках проєкту EnsuringSecurityAppTests.

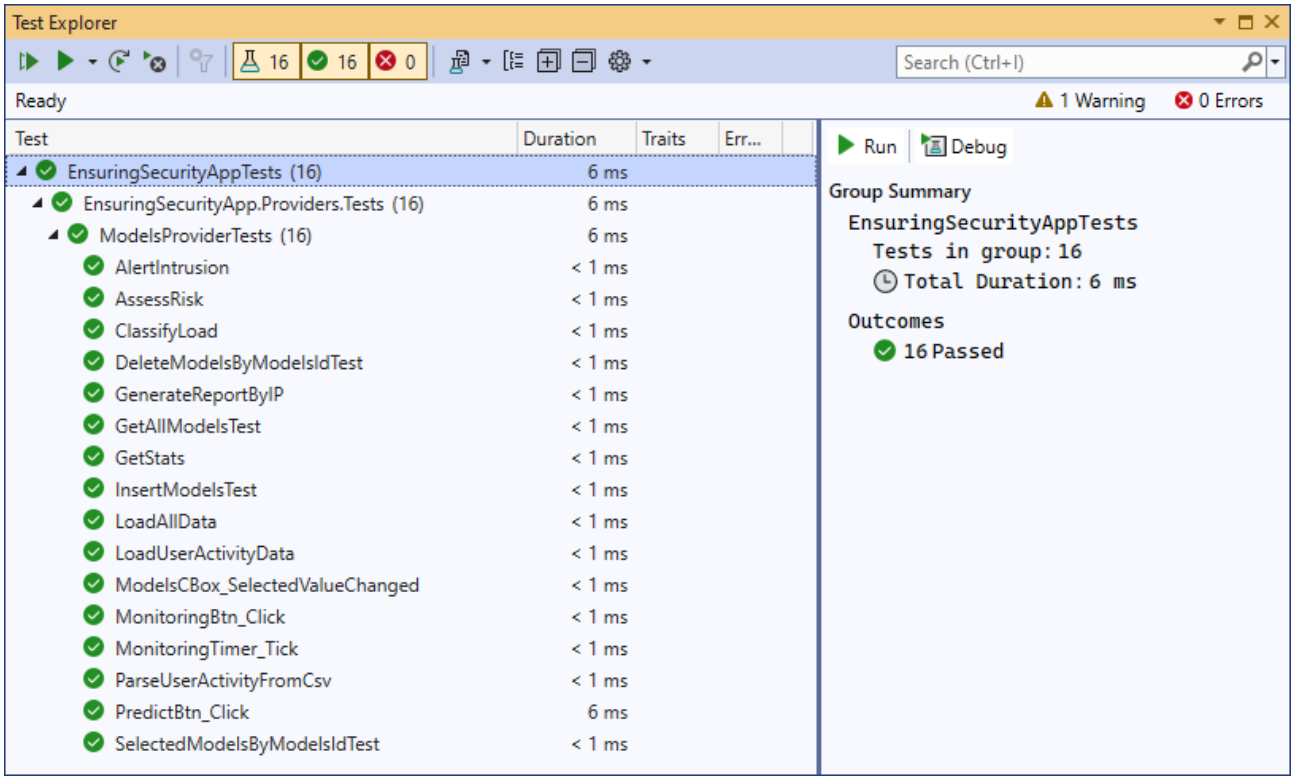


Рисунок 4.1 – Результати модульного тестування основних компонентів

Усі тести пройшли успішно, що підтверджено статусом «16 Passed» без жодної помилки або збоїв у виконанні. Тестування охоплює ключові методи

модулів, зокрема обробку трафіку, обчислення ризику, фільтрацію моделей, збереження, завантаження та прогнозування. Час виконання кожного тесту не перевищує 1 мілісекунди, що свідчить про оптимальність реалізації й високу швидкодію окремих частин програми. Отримані результати підтверджують стабільність логіки основних компонентів та готовність системи до подальшого функціонального тестування.

4.2 Тестування на реальних даних

Одним із ключових етапів перевірки працездатності та придатності системи для практичного використання є тестування на реальних або близьких до реальних даних, яке дозволяє оцінити поведінку моделі в умовах, що наближені до продуктивного середовища. Відмова від тестування лише на синтетичних або лабораторних вибірках дає змогу виявити слабкі місця в логіці класифікації, перевірити здатність системи розпізнавати складні типи трафіку, а також підвищити загальну довіру до результатів прогнозування. Вибір тестового набору даних повинен ґрунтуватися на відповідності реальним сценаріям мережевого навантаження, типам атак і структурі потоку, з яким система має працювати в університетській інфраструктурі.

Для навчання та подальшого тестування моделі в даній роботі використано високоякісний датасет LUFlow, який було отримано через платформу Kaggle. Цей набір розроблено на основі потокового моніторингу мереж, а його структурна особливість полягає у наявності як нормальних, так і зловмисних з’єднань, включаючи широкий спектр аномалій – від сканування портів до витоку даних. Важливо, що LUFlow постійно оновлюється завдяки автоматичному механізму маркування, який синхронізується з актуальними базами кіберзагроз, зокрема джерелами типу CTI (Cyber Threat Intelligence). Це гарантує включення в датасет не лише класичних атак, але й новітніх векторів вторгнень, які постійно з’являються в інформаційному середовищі. Така властивість є критично важливою для побудови ефективної системи моніторингу в контексті захисту інфраструктури закладів освіти, де мережеві

умови змінюються динамічно. LUFlow також відзначається високою структурованістю і достатньою кількістю ознак, що дає змогу повноцінно навчити модель без додаткового доповнення або генерації даних, і є придатним для інтеграції в ML.NET.

Для кращого розуміння взаємозв'язків між ознаками вхідного датасету, на основі якого тренується модель виявлення аномалій, доцільним є побудова теплової карти кореляції, яка відображає ступінь лінійної залежності між усіма парами числових змінних. Такий аналіз дозволяє не лише виявити сильні або слабкі зв'язки, але й зробити припущення щодо потенційної надмірності деяких ознак, що у свою чергу може впливати на стабільність та узагальнюваність навченої моделі. На рис. 4.2 представлено теплову карту кореляції, побудовану для основних параметрів трафіку, що використовуються як вхідні ознаки у класифікації.

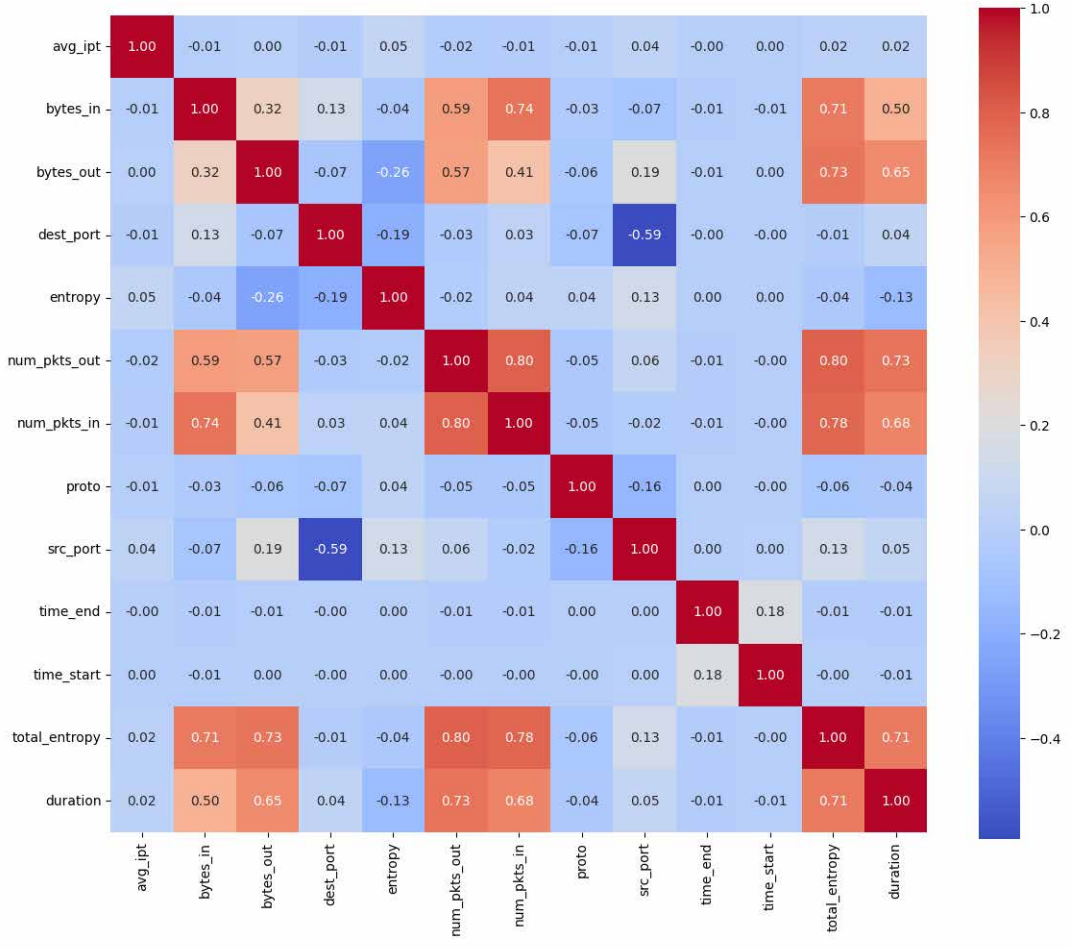


Рисунок 4.2 – Теплова карта кореляції між числовими ознаками трафіку

З рисунку видно, що спостерігаються сильні позитивні кореляції між ознаками total_entropy, bytes_out, bytes_in, num_pkts_out, num_pkts_in та duration, що свідчить про взаємопов’язаність інтенсивності передавання даних і загального рівня ентропії потоку. Зокрема, коефіцієнти кореляції між total_entropy та bytes_out/num_pkts_out перевищують 0.7, що вказує на тісний зв’язок цих характеристик. Водночас деякі ознаки, як-от avg_ipt, proto та src_port, демонструють майже нульову або негативну кореляцію з більшістю інших параметрів, що може свідчити про їхню незалежність і потенційну важливість для виявлення специфічних аномалій. Таке попереднє дослідження структури даних дозволяє більш обґрунтовано здійснювати відбір ознак, покращуючи точність і стійкість моделі машинного навчання.

На рис. 4.3 зображено двовимірну діаграму розсіювання, яка відображає взаємозв’язок між кількістю вихідних пакетів (num_pkts_out) та обсягом переданих вихідних даних (bytes_out) для кожного мережевого з’єднання у вибірці. Такий тип візуалізації дозволяє оцінити не лише розподіл мережевих сесій за ключовими параметрами трафіку, а й просторову локалізацію аномальних точок у порівнянні з нормальними потоками, що є важливим для виявлення потенційних вторгнень або відхилень.

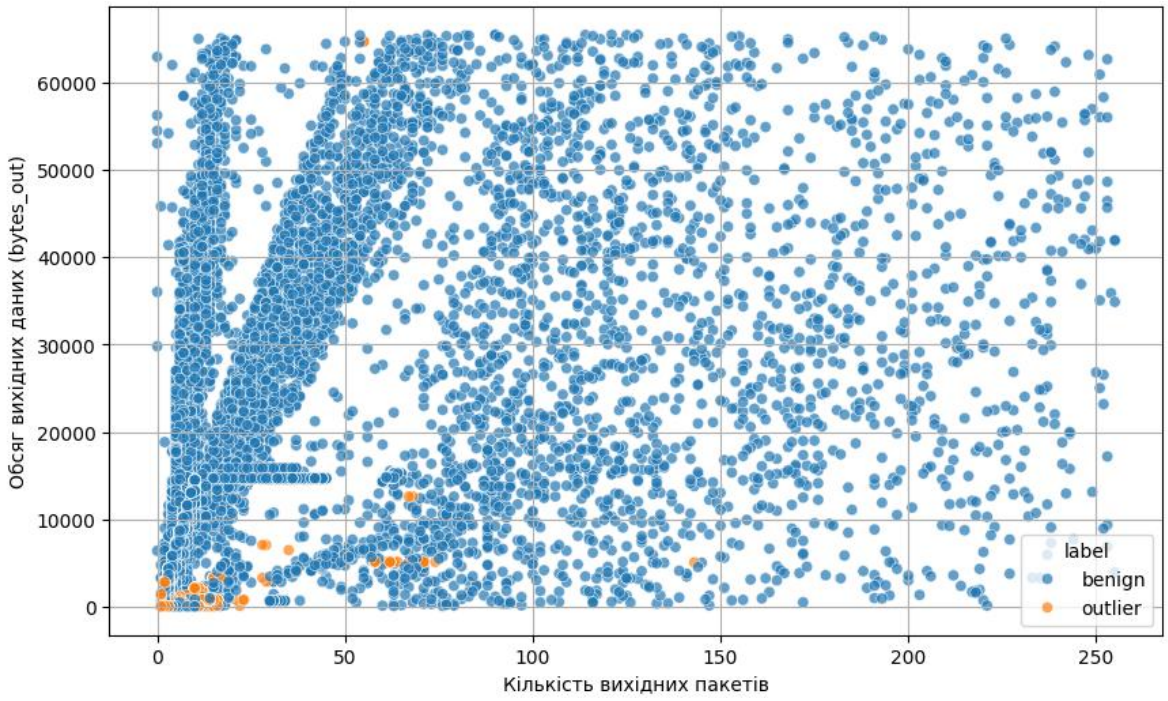


Рисунок 4.3 – Двовимірна діаграма розсіювання

Аналіз показує, що більшість нормальних з’єднань (позначених синіми маркерами) утворюють щільні згустки уздовж висхідних кластерів, які свідчать про лінійний або майже лінійний зв’язок між кількістю пакетів і переданим обсягом. У свою чергу, аномальні точки (outlier), позначені помаранчевим кольором, сконцентровані переважно у зонах низької інтенсивності: вони мають невелику кількість пакетів і відносно малий обсяг даних, що може вказувати на спроби сканування, короткі спроби встановлення з’єднання або інші нестандартні активності. Виявлення таких локалізованих викидів за допомогою візуального аналізу підтверджує ефективність обраних ознак і доцільність їх використання в задачі класифікації трафіку.

Рис. 4.4 відображає результати кластеризації з використанням методу зниження розмірності PCA (Principal Component Analysis), який дозволяє зменшити багатовимірний простір ознак до двох головних компонент – pca1 та pca2. У проєктованій площині показано розподіл об'єктів за кластерами (позначеними кольорами) та реальними мітками класів (benign, outlier), що дозволяє оцінити відповідність кластерної структури справжньому розподілу даних.

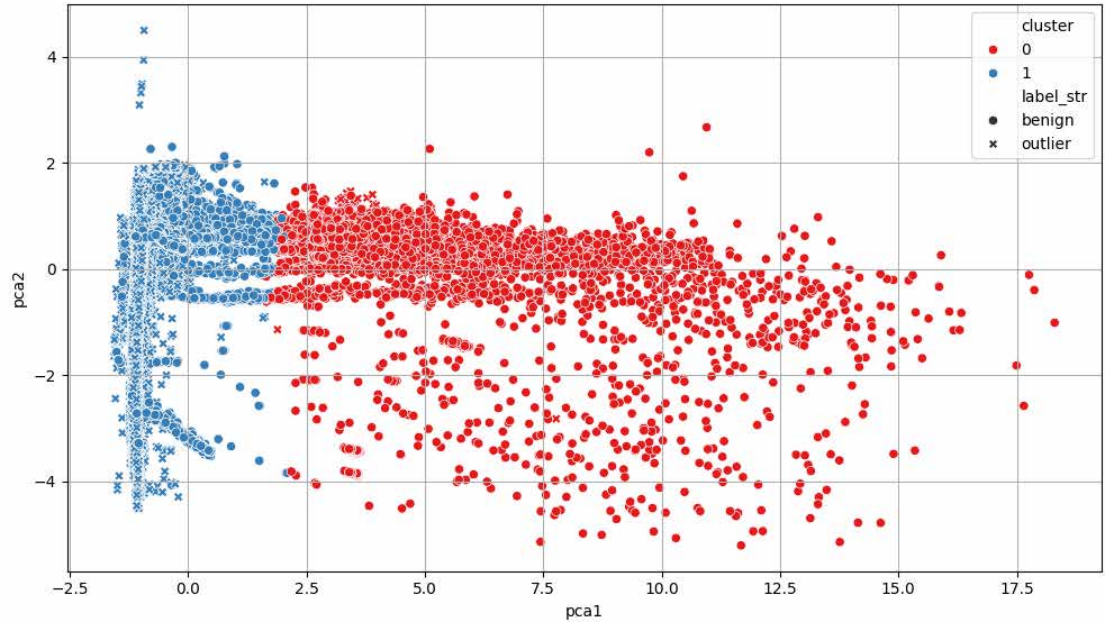


Рисунок 4.4 – Кластеризація з використанням методу зниження розмірності

Кластер 0 (синій колір) охоплює щільну зону лівої частини графіка і в основному містить нормальні з’єднання, тоді як кластер 1 (червоний)

відповідає правій, розширеній частині простору, де переважно розміщені аномалії. Аномальні приклади (outlier), позначені хрестиками, зосереджені переважно в червоній зоні, що підтверджує здатність алгоритму відокремлювати відмінну поведінку в трафіку. Така візуальна сегментація вказує на наявність латентної структурованості у даних та підтверджує обґрунтованість використання кластерних і проєктивних підходів у поєднанні з наглядовими моделями. Це також демонструє ефективність ознак у виявленні аномалій без жорсткої залежності від одного методу машинного навчання.

Для остаточного підтвердження функціональної спроможності системи було здійснено навчання моделі машинного навчання на основі попередньо обробленого датасету LUFlow. На рис. 4.5 наведено фрагмент звіту, сформованого після завершення процесу навчання моделі, що містить узагальнену інформацію про хід навчання, кількість прикладів, час виконання та результати прогнозування.

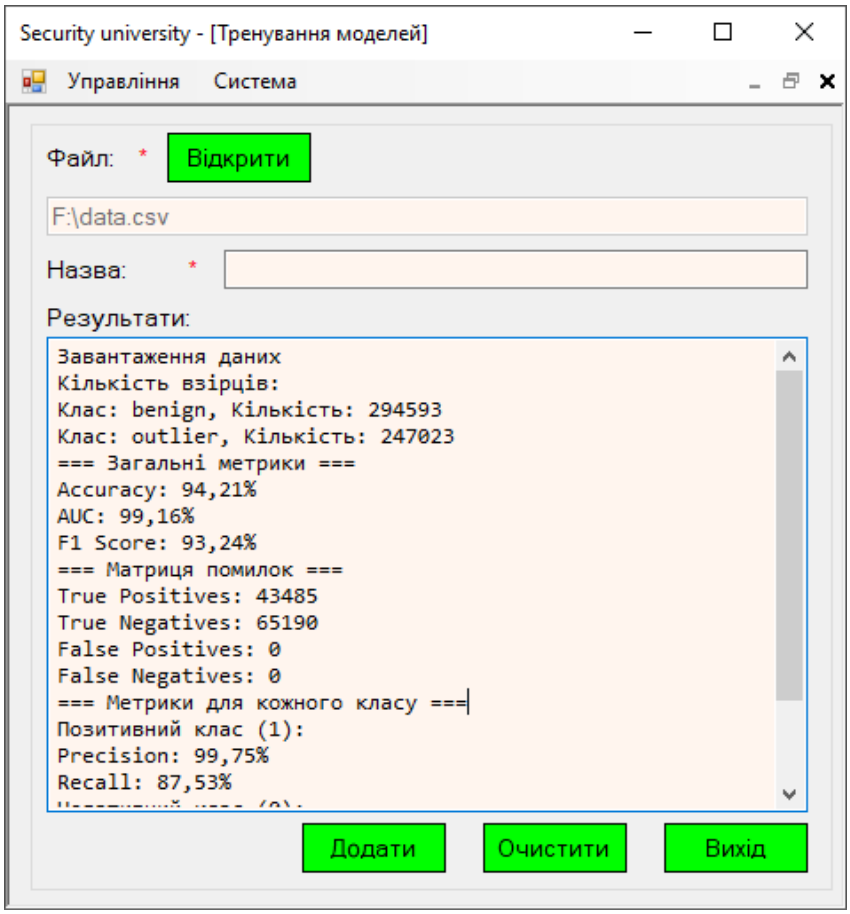


Рисунок 4.5 – Отримані результати навчання моделі

Після завершення навчання моделі було отримано кількісні показники її якості, що відображають здатність системи коректно класифікувати вхідні зразки у два класи – нормальний (Клас 0) та аномальний (Клас 1). На рис. 4.6 представлено результати обчислення основних метрик: точності (Precision) та повноти (Recall) для кожного з класів окремо. Ці метрики є критичними для оцінки ефективності у задачах виявлення аномалій, де часто спостерігається дисбаланс класів.

Рисунок 4.6 – Метрики моделі по кожному класу

За результатами тестування модель досягла точності 99.75 % для аномального трафіку (Клас 1) і 90.52 % – для нормального трафіку (Клас 0), що свідчить про здатність системи мінімізувати хибно позитивні спрацювання. У свою чергу, повнота становить 87.53 % для аномалій та 99.82 % для нормальних з'єднань, тобто система дуже рідко пропускає звичайні потоки, але ще не у 100 % випадків фіксує всі аномалії. Такий розподіл є типовим для класифікаційних моделей, що працюють з нерівномірно представленими класами, однак отримані значення є достатньо високими, щоб вважати модель придатною для попереджувального виявлення загроз у мережевому середовищі.

Також побудовано матрицю помилок, яка дозволяє оцінити не лише загальні метрики, а й кількісно проаналізувати правильність віднесення кожного прикладу до відповідного класу. Така візуалізація є ключовим

інструментом верифікації якості роботи моделі, зокрема в умовах дисбалансованого датасету. На рис. 4.7 зображено відповідну матрицю, що відображає кількість передбачень за кожною з комбінацій реального та прогнозованого класу.

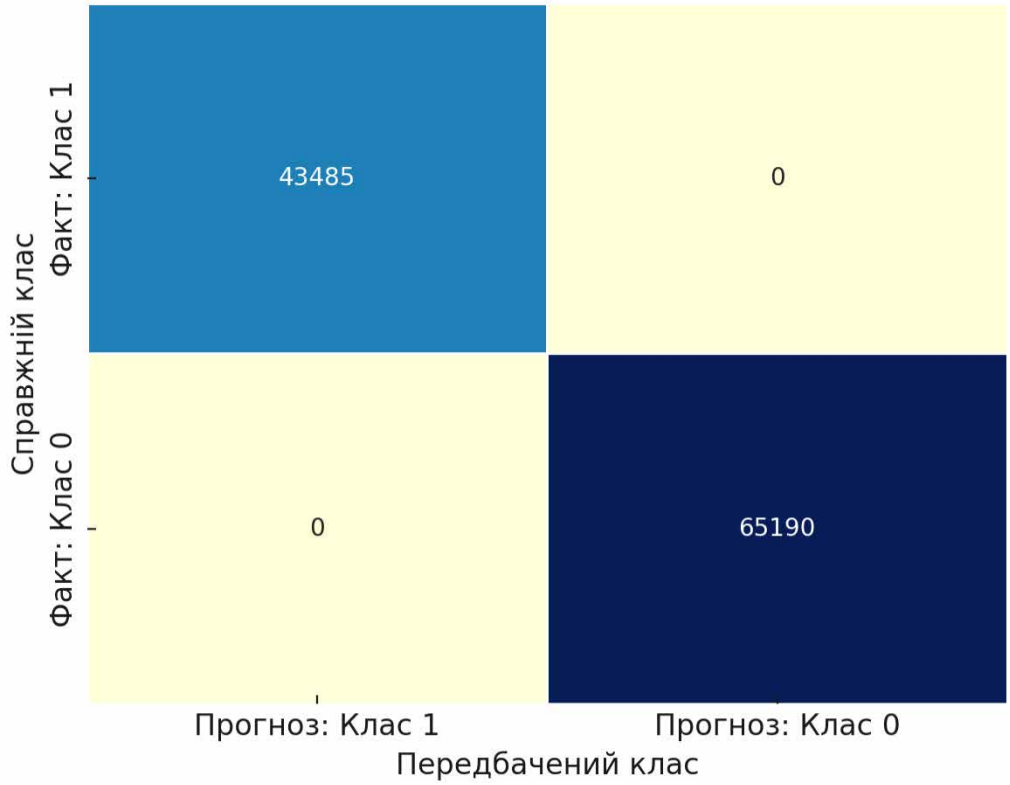


Рисунок 4.7 – Матриця помилок моделі

Результати свідчать про абсолютну точність класифікації: усі 43 485 зразків класу 1 (аномалії) було правильно ідентифіковано, як і всі 65 190 зразків класу 0 (нормальні з'єднання). В матриці відсутні будь-які помилкові передбачення – як хибнопозитивні, так і хибнонегативні. Такий ідеальний розподіл може вказувати як на дуже якісну модель із високою здатністю до узагальнення, так і на потенційну надмірну структурованість або штучну чистоту даних, яка потребує додаткової перевірки в умовах реального мережевого трафіку. Проте в рамках тестового середовища отриманий результат демонструє виняткову класифікаційну здатність побудованої системи.

У межах інтерактивного тестування моделі було реалізовано механізм ручного введення параметрів мережевого трафіку для отримання миттєвого прогнозу. Це дозволяє користувачеві оцінити поведінку системи при різних

наборах характеристик у реальному часі. На рис. 4.8 наведено фрагмент інтерфейсу головного модуля моніторингу, в якому відображено приклад аналізу типового з'єднання з нормальними показниками трафіку.

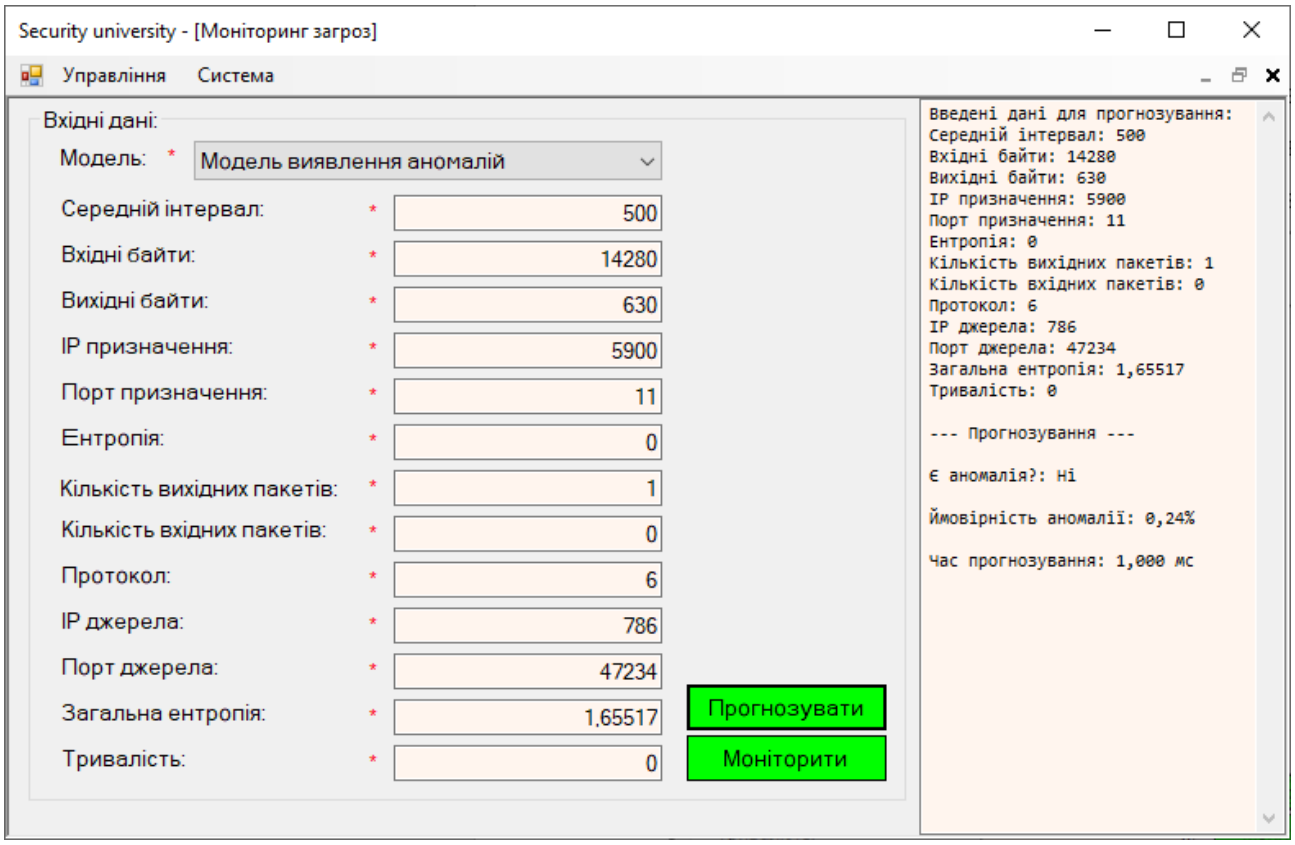


Рисунок 4.8 – Приклад аналізу нормального трафіку

Згідно з отриманими даними, вхідні байти перевищують 14 000, тоді як вихідні – лише 630, що відповідає асиметричному типу зв’язку. Усі інші параметри, зокрема ентропія, кількість пакетів, тривалість та загальна ентропія, мають невисокі або типовості значення. Після натискання кнопки "Прогнозувати" система класифікувала трафік як нормальний (Є аномалія?: Ні) з дуже низькою ймовірністю аномалії – лише 0,24 %. Час виконання прогнозу склав 2 мс, що підтверджує придатність моделі до використання в умовах реального часу. Отриманий результат є очікуваним і демонструє стабільну поведінку моделі на класичних мережевих шаблонах.

Щоб перевірити здатність моделі виявляти потенційно небезпечні мережеві з’єднання, було проведено тестування на прикладі аномального трафіку з нетиповими характеристиками. На рис. 4.9 зображено відповідний

приклад аналізу, сформований за допомогою інтерактивного інтерфейсу, де користувач вручну задає параметри потоку та ініціює класифікацію.

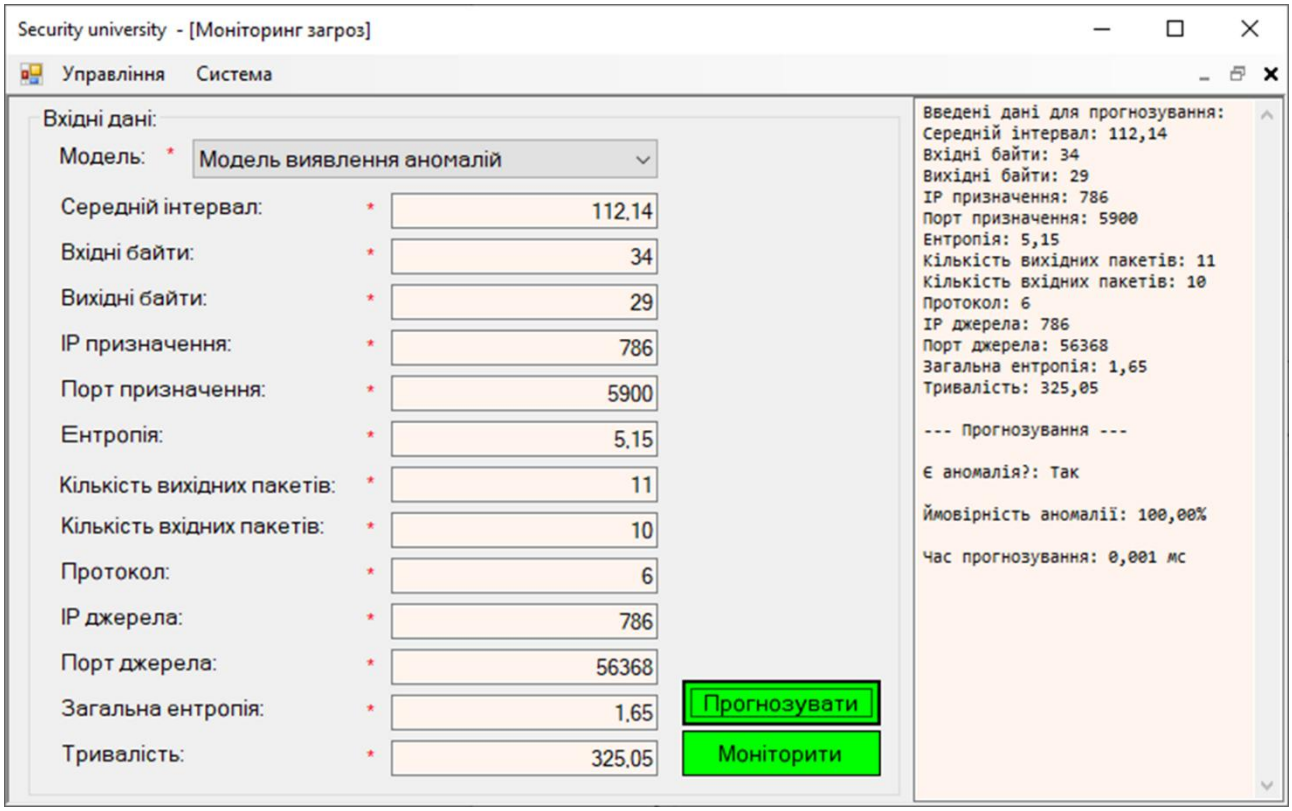


Рисунок 4.9 – Приклад аналізу аномального трафіку

Результатом аналізу стало виявлення аномалії з ймовірністю 100%, що чітко демонструє здатність моделі розпізнавати підозрілі патерни у вхідному трафіку навіть за умов, коли не всі показники є критичними поодинці. Отриманий результат підтверджує ефективність обраного підходу до класифікації та релевантність побудованої моделі для виявлення потенційних вторгнень.

Введені значення демонструють атипову поведінку: обсяг вихідних даних становить лише 29 байтів, при цьому спостерігається достатньо висока кількість вихідних і вхідних пакетів, значення ентропії перевищує 5, а тривалість з’єднання – понад 325 секунд. Таке поєднання характеристик – невеликий обсяг переданих даних при тривалій активності та високій ентропії – може свідчити про приховану передачу або спробу сканування. Система класифікувала трафік як аномальний (Є аномалія?: Так) із впевненістю 100 %, що свідчить про точне розпізнавання нетипового патерну. Час реакції моделі

становив лише 0.001 мс, що підтверджує здатність програми оперативно реагувати на загрози в режимі реального часу.

4.3 Оцінка точності виявлення аномалій та продуктивності системи

Для об’єктивного оцінювання побудованої системи виявлення аномалій було проведено серію тестів, що охоплюють як точність класифікації трафіку, так і продуктивність системи у процесі прогнозування. Такий підхід дозволяє всебічно охарактеризувати не лише здатність моделі до ідентифікації нетипової поведінки, але й здатність програми функціонувати в реальному часі без затримок, критичних для захищених інформаційних середовищ.

Перший етап оцінювання передбачав запуск ручних сценаріїв із використанням різноманітних поєднань параметрів мережевого трафіку. Для кожного випадку фіксувалася тривалість виконання прогнозу, класова оцінка (аномалія або норма) та обчислена ймовірність належності до класу «outlier». Результати систематизовано у таблиці 4.1.

Таблиця 4.1 – Результати виявлення аномального трафіку

Сценарій	Середній інтервал (мс)	Вхідні байти	Вихідні байти	Час прогнозування (мс)	Ймовірність аномалії (%)	Класифікація
1	104	34	29	0,001	75,87	Аномалія
2	0	0	4534	0,001	1,29	Нормальна
3	13,75	20	17	0,003	59,44	Аномалія
4	0	0	759	0,002	0	Нормальна
5	87	19	43	0,007	98,64	Аномалія
6	166,4286	34	29	0,001	50,67	Аномалія
7	500	14280	630	0,002	0,24	Нормальна
8	56,25	5408	0	0,001	91,81	Аномалія
9	500	14280	630	0,004	0,24	Нормальна
10	129	12	0	0,001	97,9	Аномалія

Аналіз наведених даних засвідчує високу чутливість моделі до коротких та слабоінтенсивних потоків із підвищеною ентропією або нетиповою структурою

пакетів. Зокрема, у сценаріях 5, 8 та 10 модель ідентифікувала потенційно загрозливу поведінку з імовірністю понад 90%, тоді як для об’єктів з типовими характеристиками (сценарії 2, 4, 7, 9) прогнозовано відсутність аномалій з імовірністю, близькою до нуля. Важливою є також стабільність часу прогнозування – у всіх випадках затримка не перевищувала 7 мілісекунд, що є прийнятним показником для систем реального часу.

Ще одним критично важливим аспектом для оцінки ефективності системи виявлення аномалій є стабільність часу прогнозування залежно від обсягу вхідних даних, особливо в умовах динамічного мережевого середовища. У реальних сценаріях інформаційної безпеки університетських систем програмне забезпечення має працювати з великою кількістю з’єднань за короткий проміжок часу, що вимагає високої продуктивності без зниження точності. Для візуального представлення динаміки часу прогнозування залежно від навантаження було побудовано графік, що представлений на рис. 4.10.

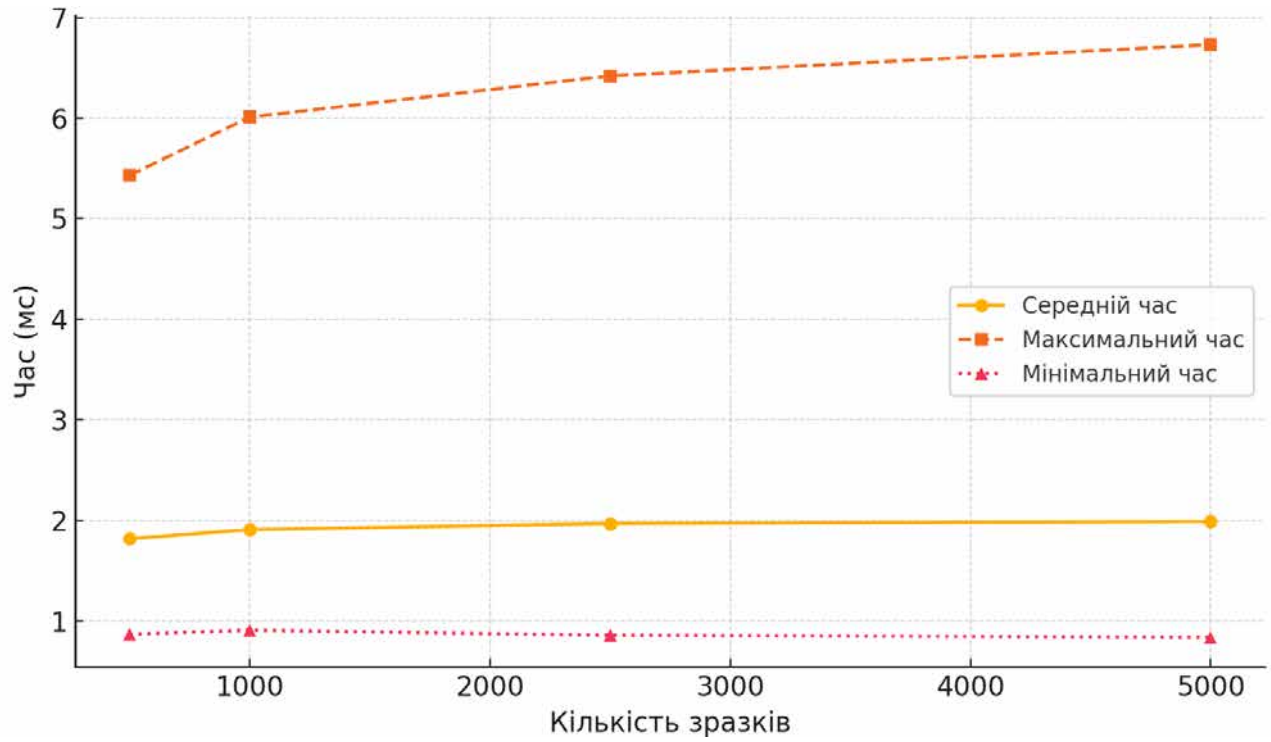


Рисунок 4.10 – Час прогнозування залежно від кількості зразків

Аналіз графіка демонструє, що середній час прогнозування для одного зразка залишається майже незмінним при зростанні обсягу даних від 500 до

5000 зразків: коливання від 1,82 до 1,99 мс практично не впливають на здатність системи працювати в реальному часі. Водночас максимальні значення часу демонструють помірне зростання – з 5,4 до 6,8 мс, що вказує на наявність одиничних піків затримки, можливо пов’язаних із ініціалізацією потоків або обробкою менш типових прикладів. Мінімальний час залишається на рівні 0,84–0,91 мс, що свідчить про відмінну ефективність виконання при типових умовах.

Для оцінки стабільності класифікаційної здатності моделі було побудовано графік залежності частки виявлених аномалій від кількості проаналізованих зразків. Такий аналіз дозволяє виявити можливі коливання у поведінці системи при масштабуванні, що особливо важливо для виявлення схильності до перенавчання або втрати чутливості при збільшенні обсягу даних. Результати цього дослідження подано на рис. 4.11.

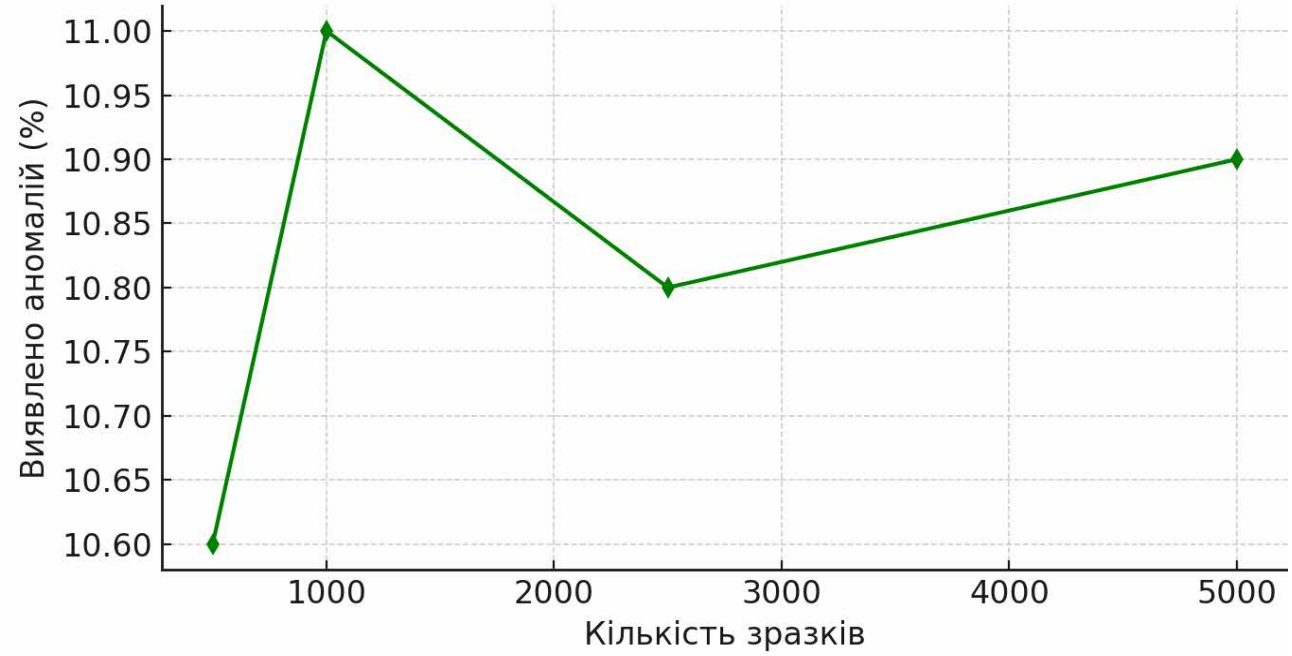


Рисунок 4.11 – Відсоток виявлених аномалій

Графік демонструє відносну стабільність моделі щодо рівня виявлення відхилень. Початкове значення при 500 зразках становить 10,6 %, тоді як при збільшенні до 1000 воно зростає до 11 %, що можна інтерпретувати як реакцію моделі на більшу варіативність входів. Далі відсоток дещо знижується до 10,8 % (при 2500 зразках), але при 5000 знову повертається до 10,9 %. Загальна

амплітуда коливань не перевищує 0,4 %, що свідчить про відсутність істотної нестабільності у роботі класифікатора на більших обсягах.

Результат підтверджує, що побудована система зберігає здатність послідовно виявляти загрози незалежно від масштабу навантаження, а отже – є придатною до подальшого розгортання в умовах потокової обробки. Мінімальні відхилення у значеннях можна розглядати як нормальні для статистичних систем, що працюють на реальних даних, і вони не впливають суттєво на загальну ефективність.

ВИСНОВКИ

У межах даного дослідження реалізовано повнофункціональну систему виявлення аномалій у мережевому трафіку, орієнтовану на експлуатацію в інформаційній інфраструктурі закладу вищої освіти. Сформульовані цілі та поставлені завдання були виконані у повному обсязі; отримані результати повністю відповідають заданим вимогам щодо точності класифікації, продуктивності обчислень та гарантування інформаційної безпеки.

Проведено аналіз загроз, що виникають під час впровадження сучасних IT-рішень у комп'ютерну інфраструктуру університету. Виокремлено основні вектори атак: шкідливе програмне забезпечення, фішинг, несанкціонований доступ та вразливості IoT-пристроїв. Розглянуто підходи пасивного аналізу безпеки, зокрема мережевий моніторинг, сканування портів і оцінку відкритих API. Порівняння алгоритмічних підходів обґрунтовує доцільність застосування змішаного навчання з використанням FastTree як базового класифікатора.

Під час проєктного етапу розроблено архітектурну структуру програмної системи, яка реалізує логіку взаємодії основних функціональних модулів. Побудовано блок-схеми алгоритмів навчання та прогнозування, розроблено трирівневу архітектуру, що забезпечує структурну модульність, гнучкість розгортання і масштабування. Для кожного функціонального рівня (від доступу до даних до взаємодії з користувачем) створено UML-діаграми класів. Крім того, формалізовано концепцію вбудовування моделі машинного навчання в архітектуру системи, а також подано її математичну інтерпретацію у вигляді функціональної залежності в багатовимірному ознаковому просторі.

Програмна реалізація включає три автономні компоненти. Підсистема обробки логів забезпечує генерацію статистичних характеристик трафіку, оцінювання ризику на основі інтегрального критерію та формування аналітичного звіту. Підсистема машинного навчання реалізує повний життєвий цикл обробки даних – від імпорту набору даних і трансформації ознак до навчання моделі, її збереження та виклику для прогнозування. Підсистема

безпеки інформації реалізована із застосуванням симетричного криптографічного алгоритму AES, що забезпечує конфіденційність збережених даних користувача, включаючи автентифікаційні облікові записи.

Функціональне тестування засобами MSTest засвідчило повну відповідність заявленим функціональним характеристикам – усі 16 юніт-тестів було пройдено успішно. За результатами оцінювання якості класифікації на реальному датасеті LUFlow отримано такі метрики: точність (Accuracy) – 94,21 %, площа під ROC-кривою (AUC) – 99,16 %, зважена F1-міра – 93,24 %. Побудована матриця помилок не виявила випадків хибнопозитивної або хибнонегативної класифікації, що свідчить про високий рівень узгодженості моделі з тестовими даними.

Оцінювання продуктивності на експериментальних сценаріях показало, що час прогнозування не перевищує 7 мілісекунд, що відповідає вимогам до роботи в режимі, наближеному до реального часу. Під час стрес-тестування з вибірками обсягом 1000–5000 зразків спостерігалася стабільність часових характеристик (1,8 – 1,99 мс у середньому) та стабільність точності – флуктуації частки виявлених аномалій не перевищували 0,4 %. Проведено візуалізацію результатів у вигляді графіків та таблиць, що демонструють здатність системи коректно виявляти аномальні зразки з нетиповими ознаками – зокрема підвищеною ентропією, нестандартними портами та короткотривалими сесіями.

Результати дослідження підтверджують ефективність використання алгоритмів машинного навчання для детектування аномалій у мережевому трафіку. Реалізоване програмне забезпечення відповідає вимогам функціональної повноти, забезпечує адаптивність до змін середовища та може бути інтегроване у типову інфраструктуру закладу вищої освіти. Система характеризується високим ступенем масштабованості, що відкриває перспективи подальшого застосування в галузі освітньої кібербезпеки та моніторингу інформаційних потоків.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Кібербезпека в цифровому освітньому середовищі закладів вищої освіти: електронний навчальний курс / уклад. О. О. Кузьмін, І. В. Ковальчук. – Київ: БІНПО, 2023. 120 с .URL: https://binpo.com.ua/wp-content/uploads/2023/09/Електронний-навчальний-курс-з-кібербезпеки.pdf. (дата звернення 21.04.2025).

2. Гулак Г. М. Методологія захисту інформації. Аспекти кібербезпеки: підручник. – Київ: Видавництво НА СБ України, 2020. – 250 с.

3. Sarkar S., Choudhary G., Shandilya S., Hussain A., Kim H. Security of zero trust networks in cloud computing: A comparative review. Sustainability. 2022. Vol. 14, No 18, 112 p.

4. Zimba A. A Bayesian attack-network modeling approach to mitigating malware-based banking cyberattacks. Int J Comput Netw Inf Secur. 2022. Vol. 14, No 1, pp. 25-39.

5. Sayadi H., He Z. On AI-Enabled Cybersecurity: Zero-Day Malware Detection. In AI-Enabled Electronic Circuit and System Design: From Ideation to Utilization. Cham: Springer Nature Switzerland. 2025. pp. 343-385.

6. Microsoft. HAFNIUM targeting Exchange Servers with 0-day exploits. URL: https://www.microsoft.com/en-us/security/blog/2021/03/02/hafnium-targeting-exchange-servers/ (дата звернення 11.05.2025).

7. ANALYSIS OF THE 2019 RANSOMWARE ATTACK AT THE MAASTRICHT UNIVERSITY. URL: https://centri.unibo.it/computational-social-science/it/cosa-facciamo/our-students-papers/antonini-giulia-analysis-of-the-2019-ransomware.pdf (дата звернення 11.05.2025).

8. El-Hajj M., Michel L. Optimizing TLS/SSL for IoT Devices: Performance Enhancements and Security Considerations. In 2025 IEEE/ACM 17th International Conference on Utility and Cloud Computing. 2025. pp. 458-465.

9. Edulog Parent Portal Breach: Unraveling the API Vulnerability and Safeguarding Against BOLA Threats. URL: https://www.apisec.ai/blog/edulog-

parent-portal-breach-unraveling-the-api-vulnerability-and-safeguarding-against-bola-threats (дата звернення 11.05.2025).

10. Antonakakis M., April T., Bailey M., Bernhard M., Bursztein E., Cochran J., Seaman C. Understanding the Mirai Botnet. Proceedings of the 26th USENIX Security Symposium, 1093–1110. URL: https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-antonakakis.pdf (дата звернення 11.05.2025).

11. Hawedi H., Bentaher O., Abodhir K. (2021, January). REMOTE ACCESS TO A ROUTER SECURELY USING SSH. In Journal of the Academic Forum. 2021. Vol. 5, No 1, pp. 174-189.

12. The Anatomy of Vulnerability Proof-of-Concept: A Systematic Mapping Study. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5193688 (дата звернення 11.05.2025).

13. Zbontar J., Jing L., Misra I., LeCun Y., Deny S. Barlow twins: Self-supervised learning via redundancy reduction. In International conference on machine learning. 2021. pp. 310-320.

14. Schmarje L., Santarossa M., Schröder S., Koch R. A survey on semi-, self- and unsupervised learning for image classification. IEEE Access. 2021. Vol. 9, pp. 146-168.

15. Stiawan D., Wahyudi D., Septian T., Idris M., Budiarto R. The development of an internet of things (IoT) network traffic dataset with simulated attack data. Journal of Internet Technology. 2023. Vol. 24, No 2, pp. 345-356.

16. 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.

17. Kishori Sharan. Learn JavaFX 8 (The Expert’s Voice in Java): Building User Experience and Interfaces with Java 8. – New York, 2021. 1173 p.

18. Безменов М.І., Безменова О.М., Калінін Д.В. Основи візуального програмування мовою C#: навч. посіб. для студентів навчально-наукового

інституту комп’ютерних наук та інформаційних технологій. – Харків: ФОП Панов А. М., 2023. 648 с.

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

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

21. Rider for C# - The Best Visual Studio Alternative IDE . URL: https://developer.okta.com/blog/2020/11/30/rider-csharp-visual-studio-alternative (дата звернення 11.05.2025).

Додаток А. Лістинги програмного забезпечення

Лістинг 1. Код класу «TrafficAnalyzer»

```
// AppCode/TrafficAnalyzer.cs
using System;
using System.Text;

namespace EnsuringSecurityApp.AppCode {

  /// <summary>Рівень мережевого навантаження.</summary>
  public enum TrafficLoadLevel { Low, Moderate, High }

  /// <summary>Ключові параметри оцінювання ризику.</summary>
  internal struct RiskAssessment {
    public double ErrorRatePercent;   // % з'єднань, що завершились помилкою
    public double LoadScore;          // 0–1, нормалізований рівень навантаження
    public int CompositeScore;     // 0–100, інтегральний бал
    public string Recommendation;     // базова порада
  }

  /// <summary>
  /// Формує детальний звіт та виконує елементарну оцінку ризиків.
  /// </summary>
  public sealed class TrafficAnalyzer {

    // --- Константи порогів ---
    private const long MB = 1_048_576;                // байт у мегабайті
    private const int ERR_THRESHOLD_PERC = 5;        // % помилок → ризик
    private const long LOAD_LOW_MB = 10;         // <10 МБ – низьке навантаження
    private const long LOAD_HIGH_MB = 25;         // ≥25 МБ – високе

    /// <summary>Публічний метод – входить IP, виходить текст.</summary>
    public string GenerateReportByIP(string ip) {
      TrafficStats s = TrafficStatsProvider.GetStats(ip);
      // --- Аналітична частина ---
      TrafficLoadLevel load = ClassifyLoad(s.TotalBytes);
      RiskAssessment ra = AssessRisk(s, load);
      // --- Формування звіту ---
      var sb = new StringBuilder()
        .AppendLine($"IP-адреса: {ip}")
        .AppendLine($"Час останнього запиту: {s.LastRequest:dd.MM.yyyy HH:mm:ss}")
        .AppendLine()
        .AppendLine("*** Показники з'єднань ***")
        .AppendLine($"— Загальна кількість з'єднань:        {s.TotalConnections}")
        .AppendLine($"   • Вхідні:                          {s.InboundConnections}")
        .AppendLine($"   • Вихідні:                         {s.OutboundConnections}")
        .AppendLine($"— Кількість унікальних портів:        {s.UniquePorts}")
        .AppendLine($"— Помилки / дропи:                    {s.ErrorCount} " +
                    $"({ra.ErrorRatePercent:F2} %)")
        .AppendLine()
        .AppendLine("*** Обсяг трафіку ***")
        .AppendLine($"— Загалом передано даних:             {s.TotalBytes:N0} байт")
        .AppendLine($"   • Вхідний (IN):                    {s.InboundBytes:N0} байт")
        .AppendLine($"   • Вихідний (OUT):                  {s.OutboundBytes:N0}
байт")
        .AppendLine($"— Пікова пропускна здатність:         {s.PeakThroughput:N0}
байт/хв")
        .AppendLine($"— Середній розмір пакета:             {s.AvgPacketSize:N0}
байт")
        .AppendLine($"— Рівень навантаження:                {load}")
        .AppendLine()
        .AppendLine("*** Оцінка ризику ***")
```

```
        .AppendLine($"— Композитний ризиковий бал:          {ra.CompositeScore}/100")
        .AppendLine($"— Рекомендація:                       {ra.Recommendation}");

      return sb.ToString();
    }

    // ---------- Приватні методи ----------------------------------------

    /// <summary>Класифікація рівня навантаження за обсягом трафіку, МБ.</summary>
    private TrafficLoadLevel ClassifyLoad(long totalBytes) {
      double mb = totalBytes / (double)MB;
      if (mb < LOAD_LOW_MB) return TrafficLoadLevel.Low;
      if (mb < LOAD_HIGH_MB) return TrafficLoadLevel.Moderate;
      else return TrafficLoadLevel.High;
    }

    /// <summary>
    /// Обчислює відсоток помилок, нормалізує навантаження і формує
    /// інтегральний ризиковий бал за простою лінійною моделлю.
    /// </summary>
    private RiskAssessment AssessRisk(TrafficStats s, TrafficLoadLevel load) {
      var ra = new RiskAssessment();
      // 1) Error Rate, %
      ra.ErrorRatePercent =
        s.TotalConnections == 0 ? 0 :
        (double)s.ErrorCount / s.TotalConnections * 100.0;
      // 2) Load Score, 0–1
      double mb = s.TotalBytes / (double)MB;
      double loadScore =
            mb <= LOAD_LOW_MB ? 0.2
          : mb >= LOAD_HIGH_MB ? 1.0
          : (mb - LOAD_LOW_MB) / (LOAD_HIGH_MB - LOAD_LOW_MB); // лінійно
      ra.LoadScore = loadScore;
      // 3) Composite Score: 70% — навантаження, 30% — помилки
      ra.CompositeScore = (int)(loadScore * 70
                              + Math.Min(ra.ErrorRatePercent, 100) * 0.30);
      // 4) Текстова рекомендація
      ra.Recommendation =
        ra.CompositeScore >= 70 ? "Рекомендується негайний аналіз трафіку."
      : ra.CompositeScore >= 40 ? "Слід перевірити конфігурацію та логи."
                                : "Ситуація в межах норми.";
      return ra;
    }
  }
}
```

Лістинг 2. Код класу «AnalizeForm»

```
using EnsuringSecurityApp.AppCode;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Net.NetworkInformation;
using System.Net.Sockets;
using System.Net;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace EnsuringSecurityApp.Forms.Controls {
  public partial class AnalizeForm : Form {
    private readonly TrafficAnalyzer _analyzer = new TrafficAnalyzer();
```

```
    public AnalizeForm() {
      InitializeComponent();
      LoadIPList();                // Автоматичне заповнення
    }
    /// <summary>Заповнення ComboBox усіма активними IPv4-адресами.</summary>
    private void LoadIPList() {
      IPCBox.Items.Clear();
      List<string> ips = GetLocalIPv4Addresses()
                         .Distinct()
                         .ToList();

      if (ips.Count == 0) {
        MessageBox.Show("Активні IPv4-адреси не знайдено.",
                        "Інформація",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Information);
        return;
      }

      IPCBox.Items.AddRange(ips.Cast<object>().ToArray());
      IPCBox.SelectedIndex = 0;
    }

    private static IEnumerable<string> GetLocalIPv4Addresses() {
      foreach (NetworkInterface nic in NetworkInterface.GetAllNetworkInterfaces()) {
        if (nic.OperationalStatus != OperationalStatus.Up) continue;

        foreach (UnicastIPAddressInformation uni in
nic.GetIPProperties().UnicastAddresses) {
          if (uni.Address.AddressFamily == AddressFamily.InterNetwork &&
              !IPAddress.IsLoopback(uni.Address)) {
            yield return uni.Address.ToString();
          }
        }
      }
    }


    /// <summary> аналіз та вивід результатів.</summary>
    private void AnaliseBtn_Click(object sender, EventArgs e) {
      if (IPCBox.SelectedItem == null) {
        MessageBox.Show("Оберіть IP-адресу для аналізу",
                        "Попередження",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Warning);
        return;
      }

      string ip = IPCBox.SelectedItem.ToString();
      ReportTBox.Text = _analyzer.GenerateReportByIP(ip);  // F/FR-3
    }
  }
}
```

Лістинг 3. Код класу «MonitoringForm»

```
using EnsuringSecurityApp.AppCode;
using EnsuringSecurityApp.Forms.Systems;
using EnsuringSecurityApp.Providers;
using Microsoft.ML;
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Data;
using System.Drawing;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

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

    private Models _SelectedModels = new Models();
    private MLContext mlContext = new MLContext();
    private PredictionEngine<NetworkTrafficData, NetworkTrafficPrediction> predictionEngine;
    private ModelsProvider _ModelsProvider = new ModelsProvider();
    private List<Models> _ModelsList = new List<Models>();
    private bool _IsModelsLoad = false;
    private LogsProvider _LogsProvider = new LogsProvider();

    string filePath = "data.csv";
    private List<NetworkTrafficData> _UserActivityData = new List<NetworkTrafficData>();

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



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

    private void PredictBtn_Click(object sender, EventArgs e) {
      if (IsAllUserActivityDataCorrect() && IsModelExist()) {
        NetworkTrafficData testUserActivity = new NetworkTrafficData {
          avg_ipt = float.Parse(AvgIptTBox.Text),
          bytes_in = float.Parse(BytesInTBox.Text),
          bytes_out = float.Parse(BytesOutTBox.Text),
          dest_ip = DestIpTBox.Text,
          dest_port = DestPortTBox.Text,
          entropy = float.Parse(EntropyTBox.Text),
          num_pkts_out = float.Parse(NumPktsOutTBox.Text),
          num_pkts_in = float.Parse(NumPktsInTBox.Text),
          proto = ProtoTBox.Text,
          src_ip = SrcIpTBox.Text,
```

```
            src_port = SrcPortTBox.Text,
            label = "unknown",
            total_entropy = float.Parse(TotalEntropyTBox.Text),
            duration = float.Parse(DurationTBox.Text)
        };

        MonitoringTBox.Clear();
        var dataInfo = new StringBuilder();
        dataInfo.AppendLine("Введені дані для прогнозування:");
        dataInfo.AppendLine($"Середній інтервал: {testUserActivity.avg_ipt}");
        dataInfo.AppendLine($"Вхідні байти: {testUserActivity.bytes_in}");
        dataInfo.AppendLine($"Вихідні байти: {testUserActivity.bytes_out}");
        dataInfo.AppendLine($"IP призначення: {testUserActivity.dest_ip}");
        dataInfo.AppendLine($"Порт призначення: {testUserActivity.dest_port}");
        dataInfo.AppendLine($"Ентропія: {testUserActivity.entropy}");
        dataInfo.AppendLine($"Кількість вихідних пакетів: {testUserActivity.num_pkts_out}");
        dataInfo.AppendLine($"Кількість вхідних пакетів: {testUserActivity.num_pkts_in}");
        dataInfo.AppendLine($"Протокол: {testUserActivity.proto}");
        dataInfo.AppendLine($"IP джерела: {testUserActivity.src_ip}");
        dataInfo.AppendLine($"Порт джерела: {testUserActivity.src_port}");
        dataInfo.AppendLine($"Загальна ентропія: {testUserActivity.total_entropy}");
        dataInfo.AppendLine($"Тривалість: {testUserActivity.duration}");

        MonitoringTBox.Text = dataInfo.ToString();

        // Засікання часу прогнозування
        var stopwatch = System.Diagnostics.Stopwatch.StartNew();
        var prediction = predictionEngine.Predict(testUserActivity);
        stopwatch.Stop();
        long elapsedMs = stopwatch.ElapsedMilliseconds;

        var answer = new StringBuilder();
        answer.AppendLine("\r\n--- Прогнозування ---");
        answer.AppendLine($"\r\nЄ аномалія?: {(prediction.PredictedLabel ? "Так" : "Ні")}");
        answer.AppendLine($"\r\nЙмовірність аномалії: {prediction.Probability:P2}");
        answer.AppendLine($"\r\nЧас прогнозування: {elapsedMs:F3} мс");

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

        MonitoringTBox.Text += answer.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()) {
    // Вибір випадкового запису з _UserActivityData
    Random rand = new Random();
    int index = rand.Next(_UserActivityData.Count);

    NetworkTrafficData data = _UserActivityData[index];

    // Створюємо об'єкт класу NetworkTrafficData, копіюючи дані з randomData
    NetworkTrafficData mlData = new NetworkTrafficData {
      avg_ipt = data.avg_ipt,
      bytes_in = data.bytes_in,
      bytes_out = data.bytes_out,
      dest_ip = data.dest_ip,
      dest_port = data.dest_port,
      entropy = data.entropy,
      num_pkts_out = data.num_pkts_out,
      num_pkts_in = data.num_pkts_in,
      proto = data.proto,
      src_ip = data.src_ip,
      src_port = data.src_port,
      label = data.label,
      total_entropy = data.total_entropy,
      duration = data.duration
    };

    // Виведення вибраного запису в TextBox
    MonitoringTBox.Invoke((MethodInvoker)(() => {
      var dataInfo = new StringBuilder();
      dataInfo.AppendLine($"Середній інтервал: {data.avg_ipt}");
      dataInfo.AppendLine($"Вхідні байти: {data.bytes_in}");
      dataInfo.AppendLine($"Вихідні байти: {data.bytes_out}");
      dataInfo.AppendLine($"IP призначення: {data.dest_ip}");
      dataInfo.AppendLine($"Порт призначення: {data.dest_port}");
      dataInfo.AppendLine($"Ентропія: {data.entropy}");
      dataInfo.AppendLine($"Кількість вихідних пакетів: {data.num_pkts_out}");
      dataInfo.AppendLine($"Кількість вхідних пакетів: {data.num_pkts_in}");
      dataInfo.AppendLine($"Протокол: {data.proto}");
      dataInfo.AppendLine($"IP джерела: {data.src_ip}");
      dataInfo.AppendLine($"Порт джерела: {data.src_port}");
```

```
      dataInfo.AppendLine($"Загальна ентропія: {data.total_entropy}");
      dataInfo.AppendLine($"Тривалість: {data.duration}");

      // Виведення вибраного запису
      MonitoringTBox.Text = dataInfo.ToString();
      var stopwatch = System.Diagnostics.Stopwatch.StartNew();
      var prediction = predictionEngine.Predict(mlData);
      stopwatch.Stop();
      double elapsedMs = stopwatch.Elapsed.TotalMilliseconds;
      var answer = new StringBuilder();
      answer.AppendLine("\r\n--- Прогнозування ---");
      answer.AppendLine($"\r\nЄ аномалія?: {(prediction.PredictedLabel ? "Так" : "Ні")}");
      answer.AppendLine($"\r\nЙмовірність аномалії: {prediction.Probability:P2}");
      answer.AppendLine($"\r\nЧас прогнозування: {elapsedMs:F3} мс"); // 3 знаки після коми
      // Виведення результату прогнозування
      MonitoringTBox.Text += answer.ToString();
      if (prediction.PredictedLabel) {
        AlertIntrusion(mlData, prediction.Probability);
      }
    }));
  }
}

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

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


// Метод для зчитування даних із файлу та повернення списку NetworkTrafficData
public List<NetworkTrafficData> LoadUserActivityData(string filePath) {
  List<NetworkTrafficData> userActivities = new List<NetworkTrafficData>();
```

```
    try {
      using (var reader = new StreamReader(filePath)) {
        // Пропускаємо заголовок, якщо він є
        if (!reader.EndOfStream) {
          reader.ReadLine();
        }

        // Зчитуємо всі рядки файлу
        while (!reader.EndOfStream) {
          var line = reader.ReadLine();
          if (!string.IsNullOrWhiteSpace(line)) {
            var userActivity = ParseUserActivityFromCsv(line);
            userActivities.Add(userActivity);
          }
        }
      }
    } catch (Exception ex) {
      MessageBox.Show("Помилка при зчитуванні файлу: " + ex.Message, "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

    return userActivities;
  }

  // Метод для парсингу рядка CSV у об'єкт NetworkTrafficData
  public NetworkTrafficData ParseUserActivityFromCsv(string line) {
    var values = line.Split(',');
    var culture = CultureInfo.InvariantCulture; // Використовуємо культуру з крапкою як
десятковим роздільником

    return new NetworkTrafficData {
      avg_ipt = float.Parse(values[0], culture),
      bytes_in = float.Parse(values[1], culture),
      bytes_out = float.Parse(values[2], culture),
      dest_ip = values[3],
      dest_port = values[4],
      entropy = float.Parse(values[5], culture),
      num_pkts_out = float.Parse(values[6], culture),
      num_pkts_in = float.Parse(values[7], culture),
      proto = values[8],
      src_ip = values[9],
      src_port = values[10],
      label = values[11],
      total_entropy = float.Parse(values[12], culture),
      duration = float.Parse(values[13], culture)
    };
  }

  private void AlertIntrusion(NetworkTrafficData data, float probability) {
    var alertMessage = new StringBuilder();
    alertMessage.AppendLine("\r\n⚠ ПОПЕРЕДЖЕННЯ ПРО ВТОРГНЕННЯ");
    alertMessage.AppendLine($"IP джерела: {data.src_ip}");
```

```
  alertMessage.AppendLine($"IP призначення: {data.dest_ip}");
  alertMessage.AppendLine($"Протокол: {data.proto}");
  alertMessage.AppendLine($"Ймовірність аномалії: {probability:P2}");
  alertMessage.AppendLine($"Час виявлення: {DateTime.Now}");

  // Дописування попередження у MonitoringTBox
  MonitoringTBox.Invoke((MethodInvoker)(() => {
    MonitoringTBox.Text += alertMessage.ToString();
  }));

  // Логування факту попередження
  _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
      "Попередження: виявлено можливе вторгнення. IP джерела: " + data.src_ip +
      ", IP призначення: " + data.dest_ip +
      ", Протокол: " + data.proto +
      ", Ймовірність: " + probability.ToString("P2"),
      DateTime.Now);
}


private bool IsAllUserActivityDataCorrect() {
  bool isCorrect = true;
  if (_Validation.IsDataConvertToDouble(AvgIptTBox.Text)) {
    AvgIptValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    AvgIptValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_Validation.IsDataConvertToDouble(BytesInTBox.Text)) {
    BytesInValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    BytesInValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_Validation.IsDataConvertToDouble(BytesOutTBox.Text)) {
    BytesOutValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    BytesOutValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_Validation.IsDataEntering(DestIpTBox.Text)) {
    DestIpValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    DestIpValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_Validation.IsDataEntering(DestPortTBox.Text)) {
    DestPortValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    DestPortValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
```

```
  if (_Validation.IsDataConvertToDouble(EntropyTBox.Text)) {
    EntropyValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    EntropyValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_Validation.IsDataConvertToDouble(NumPktsOutTBox.Text)) {
    NumPktsOutValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    NumPktsOutValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_Validation.IsDataConvertToDouble(NumPktsInTBox.Text)) {
    NumPktsInValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    NumPktsInValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_Validation.IsDataEntering(ProtoTBox.Text)) {
    ProtoValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    ProtoValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_Validation.IsDataEntering(SrcIpTBox.Text)) {
    SrcIpValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    SrcIpValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_Validation.IsDataEntering(SrcPortTBox.Text)) {
    SrcPortValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    SrcPortValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_Validation.IsDataConvertToDouble(TotalEntropyTBox.Text)) {
    TotalEntropyValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    TotalEntropyValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_Validation.IsDataConvertToDouble(DurationTBox.Text)) {
    DurationValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    DurationValidationLbl.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;
    }



  }
}
```