ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система моніторингу інфраструктури на основі штучного інтелекту»

на здобуття освітнього ступеня магістра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

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

_____ Богдан Горовий
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Богдан Горовий
група КНДМ-63

Керівник: Сергій ПРОКОПОВ
науковий ступінь, вчене звання к.т.н., доцент

Рецензент:
науковий ступінь, вчене звання _____
(Ім'я, ПРІЗВИЩЕ)

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Магістр

Спеціальність Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Горовому Богдану Володимировичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Система моніторингу інфраструктури на основі штучного інтелекту»

керівник кваліфікаційної роботи Сергій ПРОКОПОВ к.т.н., доцент,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024р. №320

2. Строк подання кваліфікаційної роботи «15» грудня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, методи машинного навчання та аналізу даних, методи розробки програмного забезпечення.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

4.1. Теоретичні основи застосування штучного інтелекту в системі моніторингу інфраструктури.
4.2. Вибір технологічного стеку та опис методології розробки.
4.3. Розробка системи виявлення атак на інфраструктуру за допомогою штучного інтелекту.

5. Перелік графічного матеріалу: *презентація*
 5.1 Системи моніторингу та їх методи обробки даних.
 5.2 Технології, інструменти та методологія для розробки системи.
 5.3 Проектування системи.
 5.4 Тренування моделі штучного інтелекту.
6. Дата видачі завдання «16» вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.09-27.09.24	Виконано
2	Аналіз методів обробки даних	30.09-11.10.24	Виконано
3	Проектування та підбір технологій реалізації	14.10-25.10.24	Виконано
4	Розробка системи моніторингу	28.10-08.11.24	Виконано
5	Завантаження та обробка датасету для тренування моделі	11.11-22.12.24	Виконано
6	Тренування моделі та оцінка	25.11-29.11.24	Виконано
7	Оформлення роботи: вступ, висновки, реферат	02.12-06.12.24	Виконано
8	Розробка демонстраційних матеріалів	09.12-13.12.24	Виконано

Здобувач вищої освіти _____ Богдан ГОРОВИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Сергій ПРОКОПОВ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 106 стор., 5 табл., 42 рис., 46 джерел.

Наукове завдання – розробка системи моніторингу інфраструктурних об’єктів з використанням штучного інтелекту.

Мета роботи – підвищити рівень безпеки і надійності критичних інфраструктурних систем.

Об’єкт дослідження – процес виявлення та класифікації аномалій у мережевому трафіку інфраструктури з використанням методів машинного навчання.

Предмет дослідження – система моніторингу інфраструктурних об’єктів з використанням штучного інтелекту.

Методи дослідження – аналіз існуючих систем моніторингу, методи інтелектуальної обробки даних, застосування алгоритмів машинного навчання, а також тестування ефективності системи на контрольних сценаріях.

Короткий зміст роботи: Проведено огляд сучасних технологій і методів, що застосовуються в системах моніторингу інфраструктури.

Розроблено систему, здатну виявляти атаки на основі багатокласової класифікації. Система виявилася здатною адаптуватися до змін у роботі інфраструктури, що забезпечує ефективне реагування на різноманітні загрози. Під час тестування на реальних датасетах було досягнуто високої точності у виявленні аномалій, яка перевищує 90%, що свідчить про значне підвищення ефективності моніторингу.

КЛЮЧОВІ СЛОВА: МОНІТОРИНГ ІНФРАСТРУКТУРИ, МАШИННЕ НАВЧАННЯ, АНОМАЛІЇ, КЛАСИФІКАЦІЯ, ШТУЧНИЙ ІНТЕЛЕКТ, МЕРЕЖЕВИЙ ТРАФІК.

ABSTRACT

Text part of the master's qualification work: 106 pages, 42 pictures, 5 table, 46 sources.

Scientific task is to develop a system for monitoring infrastructure facilities using artificial intelligence.

Goal of the work is to improve the safety and reliability of critical infrastructure systems.

Object of research – the process of detecting and classifying anomalies in infrastructure network traffic using machine learning methods.

Subject of research – a system for monitoring infrastructure objects using artificial intelligence.

Research methods – analysis of existing monitoring systems, methods of intelligent data processing, application of machine learning algorithms, as well as testing the system's effectiveness on control scenarios.

Summary of the work: A review of modern technologies and methods used in infrastructure monitoring systems is carried out.

A system capable of detecting attacks based on multi-class classification has been developed. The system proved to be able to adapt to changes in infrastructure operation, which ensures an effective response to various threats. During testing on real datasets, a high accuracy in detecting anomalies exceeding 90% was achieved, which indicates a significant increase in monitoring efficiency.

KEYWORDS: INFRASTRUCTURE MONITORING, MACHINE LEARNING, ANOMALIES, CLASSIFICATION, ARTIFICIAL INTELLIGENCE, NETWORK TRAFFIC.

ЗМІСТ

ВСТУП

Сучасні інфраструктурні об’єкти, такі як транспортні мережі, енергетичні системи, водопостачання та комунальні послуги, є невід’ємною складовою функціонування суспільства та економіки. Їх безперебійна робота безпосередньо впливає на ефективність виробництва, якість життя населення та забезпечення національної безпеки. Проте, в умовах постійного зростання складності інфраструктурних систем, а також підвищення навантаження на них, все частіше постають проблеми, пов’язані з підтримкою їх надійності, оперативністю виявлення несправностей та прогнозуванням можливих загроз.

Традиційні методи моніторингу інфраструктури, які передбачають періодичні інспекції, автоматизований збір даних від сенсорів та оперативне реагування на аварії, не завжди дозволяють своєчасно виявляти загрози та попереджати катастрофічні наслідки. З огляду на це, застосування нових технологій, зокрема штучного інтелекту (ШІ), відкриває значні перспективи для вирішення цих проблем. [1]

ШІ пропонує нові інструменти для аналізу великих обсягів даних, зібраних з інфраструктурних об’єктів, виявлення прихованих закономірностей та аномалій, а також прогнозування можливих несправностей або загроз у реальному часі. Використання методів машинного навчання дозволяє адаптувати системи моніторингу до мінливих умов експлуатації, швидше реагувати на зміни та забезпечувати ефективне управління ресурсами. Водночас, ШІ дозволяє підвищити точність оцінки стану об’єктів інфраструктури, що особливо важливо для великих та складних систем, таких як транспортні мережі або енергетичні комплекси [2].

Актуальність даного дослідження обумовлена не лише технічними викликами, а й економічними та соціальними аспектами. Зокрема, в Україні питання модернізації та оптимізації інфраструктури є надзвичайно актуальним у контексті підвищення енергоефективності, розвитку транспортної системи та забезпечення стабільної роботи життєво важливих систем у кризових умовах.

Впровадження технологій штучного інтелекту у системи моніторингу інфраструктури сприятиме не лише зниженню ризиків виникнення аварій, але й оптимізації витрат на їх експлуатацію та ремонт. Крім того, сучасна світова тенденція до цифровізації та автоматизації процесів вимагає від України інтеграції таких рішень для забезпечення конкурентоспроможності на глобальному рівні.

Критичний аналіз наявних рішень показує, що існуючі підходи до моніторингу інфраструктур часто обмежуються використанням базових автоматизованих систем контролю, які мають обмежену здатність до самонавчання і не можуть ефективно працювати в умовах великої кількості змінних параметрів. Саме тому дослідження застосування штучного інтелекту для підвищення ефективності моніторингу інфраструктури є актуальним та перспективним напрямком, який сприятиме підвищенню надійності та безпеки ключових об’єктів, а також зменшенню ризиків порушення їхньої роботи, що може негативно вплинути на стабільність суспільних і економічних процесів.

Об’єктом дослідження є процес виявлення та класифікації аномалій у мережевому трафіку інфраструктури з використанням методів машинного навчання.

Предметом дослідження є методи застосування штучного інтелекту, зокрема алгоритми машинного навчання, для виявлення загроз та аномалій в інфраструктурних системах на основі аналізу великих обсягів даних, зібраних у реальному часі.

Мета даної роботи полягає у розробці системи моніторингу інфраструктурних об’єктів з використанням штучного інтелекту для виявлення потенційних загроз та атак, що сприятиме підвищенню безпеки і надійності критичних інфраструктурних систем.

Основні задачі, які необхідно вирішити для досягнення поставленої мети:

- провести огляд сучасних технологій і методів, що застосовуються в системах моніторингу інфраструктури, і визначити їхні ключові особливості та недоліки;

- визначити найбільш ефективні методи обробки даних та застосування машинного навчання для моніторингу інфраструктурних об’єктів, включаючи аналіз класифікаційних методів;

- обґрунтувати вибір технологічного стеку та інструментів для реалізації системи моніторингу з використанням методів штучного інтелекту для виявлення кіберзагроз;

- розробити систему, здатну виявляти атаки на основі багатокласової класифікації, оцінити її ефективність у реальних умовах, з використанням реальних датасетів для тренування моделі.

Наукова новизна одержаних результатів полягає у розробці підходу до виявлення типу кібернетичної атаки на інфраструктурні об’єкти з використанням алгоритму машинного навчання максимального ентропійного методу з обмеженою пам’яттю Брёйдена-Флетчера-Гольдфарба-Шанно. Запропоноване рішення дозволяє підвищити точність класифікації атак в реальному часі, що забезпечує більш ефективний аналіз великих масивів даних, зібраних від систем Інтернету речей (IoT). Використання цього методу відкриває нові можливості для моніторингу критичних інфраструктур і своєчасного виявлення потенційних загроз.

Практичне значення одержаних результатів полягає у можливості застосування розробленої системи для виявлення кіберзагроз в реальних умовах. Система дозволяє не лише ідентифікувати тип атаки за допомогою алгоритму максимального ентропійного методу з обмеженою пам’яттю, але й ефективно застосовувати її в системах моніторингу, що використовують IoT-сенсори. Це сприяє підвищенню рівня безпеки інфраструктурних об’єктів, зменшенню часу реакції на інциденти та поліпшенню загальної захищеності систем від можливих кіберзагроз.

1 ТЕОРЕТИЧНІ ОСНОВИ ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ В СИСТЕМІ МОНІТОРИНГУ ІНФРАСТРУКТУРИ

1.1 Системи моніторингу інфраструктури

Системи моніторингу інфраструктури є невід'ємною частиною забезпечення надійної роботи різноманітних об'єктів критичної інфраструктури, таких як енергетичні мережі, транспортні системи, водопостачання, комунальні послуги, і багато інших. Основна мета цих систем полягає у безперервному контролі стану об'єктів, швидкому виявленні відхилень у їхній роботі та своєчасному реагуванні на виникаючі загрози, щоб мінімізувати ризики збою, пошкодження або порушення роботи.

Сучасні системи моніторингу активно використовують дані, зібрані з різноманітних сенсорів та пристроїв IoT, для отримання інформації про поточний стан інфраструктурних об'єктів. Завдяки цьому забезпечується реальний час для прийняття рішень та швидке реагування на критичні ситуації. Моніторинг охоплює широкий спектр аспектів: від контролю фізичних параметрів об'єктів (температури, тиску, вологості) до аналізу трафіку в мережах, виявлення аномалій та оцінки рівня загроз.

Окрім технічних характеристик, системи моніторингу також відіграють важливу роль у захисті інфраструктур від кіберзагроз. Зростання цифровізації й інтеграція IoT-систем відкривають нові можливості для атак на критичні об'єкти, тому моніторинг таких систем потребує використання передових підходів до обробки великих масивів даних і виявлення потенційних загроз [3].

Традиційні системи моніторингу часто базуються на попередньо визначених правилах, що обмежує їхню здатність адаптуватися до нових типів загроз або нестандартних ситуацій. Для вирішення цієї проблеми все частіше використовуються методи машинного навчання та штучного інтелекту, які дозволяють системам моніторингу не лише аналізувати дані, але й навчатися на основі нових патернів поведінки і аномалій.

Впровадження штучного інтелекту в системи моніторингу інфраструктури дозволяє підвищити ефективність аналізу даних, автоматизувати процеси виявлення загроз і аномалій, а також оптимізувати процеси управління інфраструктурними об’єктами.

На ринку представлено багато систем, які виконують функції моніторингу мережевих і критичних інфраструктур. Більшість із них орієнтовані на забезпечення безперервного контролю за станом систем, виявлення аномалій, аналіз трафіку і захист від потенційних кіберзагроз [4].

Nagios XI – це програмне забезпечення для моніторингу IT-інфраструктури, яке дозволяє контролювати стан мережевих пристроїв, серверів, додатків та сервісів (рис. 1.1). Його основне призначення полягає в оперативному виявленні проблем, забезпеченні безперервної роботи систем та швидкому реагуванні на збої. Nagios XI надає централізовану панель для моніторингу і дозволяє отримувати сповіщення в реальному часі про стан інфраструктури, що допомагає запобігати зупинкам у роботі бізнес-критичних процесів.

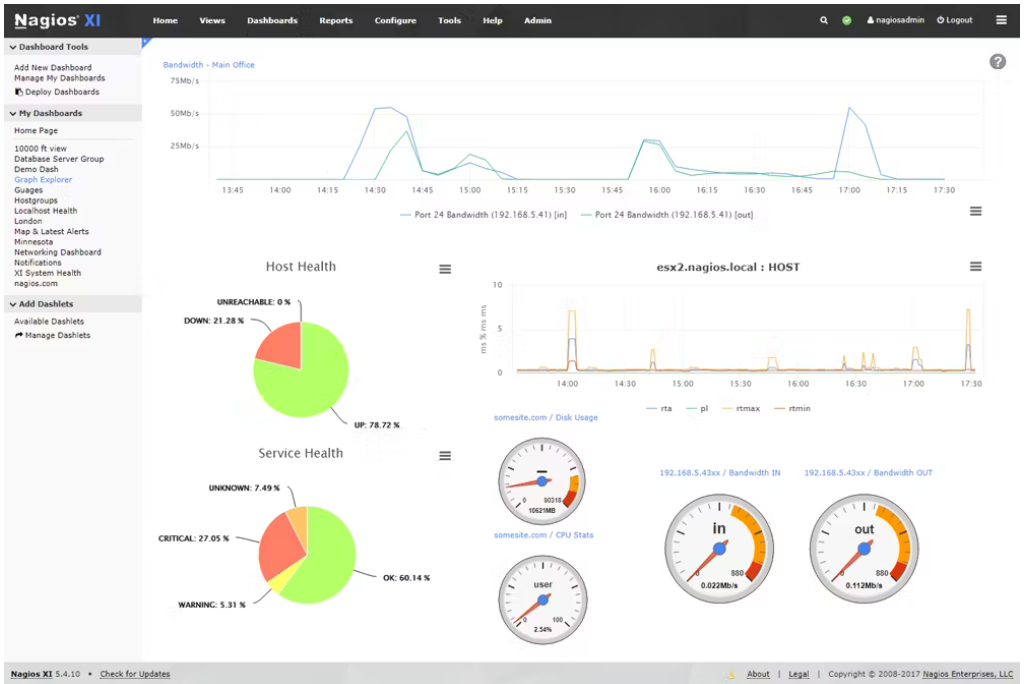


Рисунок 1.1 – Приклад інтерфейсу системи «Nagios XI» [5]

Архітектура Nagios XI побудована на основі кількох ключових компонентів, включаючи ядро Nagios Core, яке відповідає за основну функціональність моніторингу. Додаткові модулі та плагіни дозволяють розширювати можливості

системи для моніторингу різних типів пристроїв та сервісів. Інтерфейс користувача надає гнучкі інструменти для візуалізації даних, конфігурації моніторингу та управління інфраструктурою. Окрім того, Nagios XI підтримує інтеграцію з іншими IT-системами для покращення автоматизації процесів управління інфраструктурою.

Переваги Nagios XI:

- гнучка модульна архітектура. Можливість додавання плагінів і розширень для моніторингу будь-яких типів пристроїв та сервісів;

- широкі можливості інтеграції. Підтримка інтеграції з іншими IT-системами (CMDB, ERP), що дозволяє автоматизувати процеси управління інфраструктурою;

- централізоване сповіщення. Система оперативного сповіщення про збої дозволяє швидко реагувати на проблеми та мінімізувати час простоїв;

- візуалізація та звітність. Зручні панелі моніторингу та звіти надають повний огляд стану системи в реальному часі.

Недоліки Nagios XI:

- складність налаштування. Початкова конфігурація може бути складною для нових користувачів, вимагає значного часу для навчання та налаштування;

- відсутність автоматичного виявлення пристроїв. Ручне додавання мережевих елементів займає час і може викликати помилки;

- високі вимоги до ресурсів. При моніторингу великих інфраструктур система може споживати значні обчислювальні ресурси;

- застарілий інтерфейс користувача. Деякі користувачі вважають, що інтерфейс недостатньо сучасний порівняно з іншими рішеннями [6].

Отже, Nagios XI є потужним інструментом для моніторингу IT-інфраструктури, який надає гнучкі можливості налаштування і інтеграції з іншими системами. Його головними перевагами є модульна архітектура, широкі

можливості інтеграції та централізоване сповіщення про проблеми. Однак, початкова складність налаштування та високі вимоги до ресурсів можуть обмежити використання для менш досвідчених користувачів або у невеликих організаціях.

ManageEngine OpManager – це рішення для моніторингу мережі та управління IT-інфраструктурою, призначене для моніторингу мережевих пристроїв, серверів, додатків і віртуальних середовищ (рис. 1.2). Його основна функція полягає у забезпеченні безперебійної роботи мережі шляхом своєчасного виявлення збоїв і моніторингу продуктивності. Система надає користувачам можливість отримувати вичерпну інформацію про стан IT-ресурсів та запобігати простоям за допомогою оперативних сповіщень та автоматизації процесів.

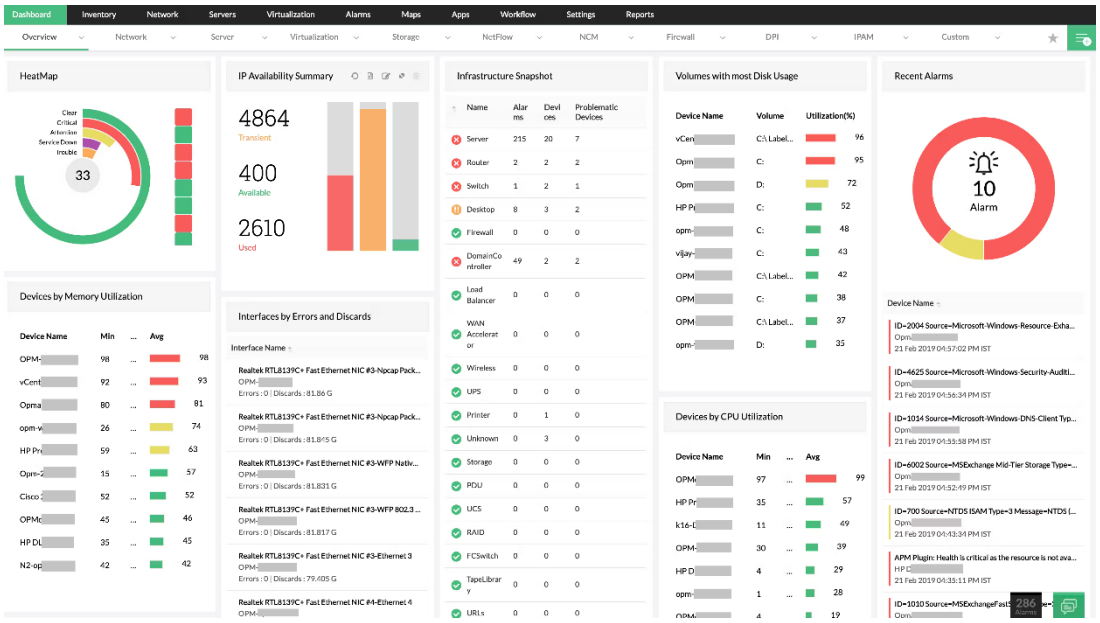


Рисунок 1.2 – Приклад інтерфейсу системи «ManageEngine OpManager» [7]

Архітектура ManageEngine OpManager базується на клієнт-серверній моделі з використанням центрального сервера для збору та аналізу даних. Сервер здійснює моніторинг за допомогою агентів або агентless методів, використовуючи протоколи SNMP, WMI та ICMP для збору даних про стан мережі. Система інтегрується з іншими продуктами ManageEngine, що дозволяє легко розширювати її функціональні можливості, наприклад, для управління журналами подій або автоматизації IT-процесів. Інтерфейс надає можливості для візуалізації даних, створення звітів та проведення аналізу продуктивності у реальному часі.

Переваги ManageEngine OpManager:

- інтуїтивно зрозумілий інтерфейс. Користувачам легко працювати завдяки зручній панелі управління, яка дозволяє швидко отримувати доступ до основних функцій;

- підтримка широкого спектру протоколів. Система підтримує SNMP, WMI, ICMP та інші протоколи для збирання даних, що забезпечує моніторинг різних типів пристроїв без потреби встановлення агентів;

- масштабованість. Рішення добре підходить як для малих, так і для великих мереж завдяки можливості моніторингу тисяч пристроїв;

- автоматизація завдань. Підтримка сценаріїв для автоматизації процесів, таких як усунення неполадок або виконання певних дій при виникненні помилок.

Недоліки ManageEngine OpManager:

- висока вартість ліцензії. Для великих мереж вартість ліцензії може бути значною, що робить його недоступним для менших компаній;

- великі вимоги до ресурсів. Система потребує значної кількості ресурсів для обробки даних та моніторингу великих інфраструктур;

- обмеження в налаштуваннях сповіщень. Налаштування сповіщень можуть бути недостатньо гнучкими для складних мереж або специфічних вимог;

- затримки у підтримці клієнтів. Деякі користувачі відзначають повільну реакцію технічної підтримки, що може вплинути на вирішення критичних проблем [8].

Отже, ManageEngine OpManager – це потужне рішення для моніторингу мереж, яке пропонує інтуїтивний інтерфейс, підтримку багатьох протоколів і можливості автоматизації. Однак, його вартість та вимоги до ресурсів можуть стати проблемою для невеликих компаній або організацій з обмеженим бюджетом. Незважаючи на деякі недоліки, система є високоефективною для масштабного моніторингу та управління складними IT-інфраструктурами.

Paessler PRTG – це система моніторингу, призначена для контролю мережевої інфраструктури, серверів, додатків та інших IT-ресурсів у реальному часі (рис. 1.3). Основне призначення PRTG полягає у відстеженні стану мережевих пристроїв, виявленні аномалій, вимірюванні продуктивності та забезпеченні безперервної роботи IT-систем шляхом своєчасного реагування на проблеми.

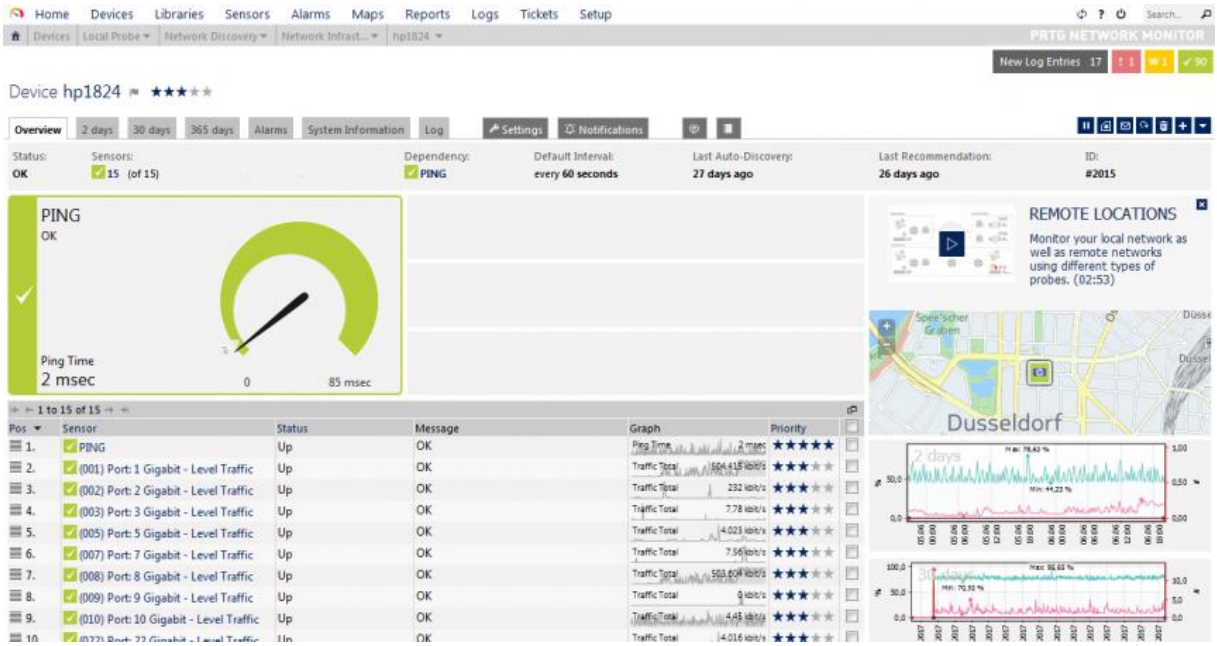


Рисунок 1.3 – Приклад інтерфейсу системи «Paessler PRTG» [9]

Архітектура Paessler PRTG побудована за принципом клієнт-серверної моделі, де центральний сервер здійснює збір даних з різних мережевих пристроїв за допомогою різних методів, таких як SNMP, WMI, NetFlow, sFlow, та інших. Дані аналізуються на сервері та відображаються через веб-інтерфейс або спеціальний клієнт. PRTG також підтримує розподілену архітектуру з використанням додаткових проб для моніторингу віддалених локацій. Система надає вбудовані інструменти для налаштування оповіщень та створення детальних звітів, що дозволяє ефективно керувати IT-інфраструктурою з єдиної консолі.

Переваги Paessler PRTG:

- універсальність моніторингу. Підтримує широкий спектр протоколів (SNMP, WMI, NetFlow, sFlow та інші), що дозволяє контролювати різноманітні пристрої та сервіси в мережі;

- легка інтеграція та налаштування. Інтуїтивно зрозумілий інтерфейс та швидкий процес встановлення, що дозволяє користувачам без проблем почати моніторинг систем;

- вбудована система сповіщень. Налаштування автоматичних сповіщень, які надходять через електронну пошту, SMS або інші канали, що дозволяє своєчасно реагувати на проблеми;

- можливість розподіленого моніторингу. Підтримка віддалених проб дозволяє контролювати різні локації без втрати ефективності, що робить його ідеальним для великих підприємств із розподіленою інфраструктурою.

Недоліки Paessler PRTG:

- обмеження у безкоштовній версії. Безкоштовна версія обмежена до 100 датчиків, що може бути недостатнім для середніх та великих мереж;

- високі вимоги до ресурсів. Під час моніторингу великої кількості датчиків система може споживати значні обчислювальні ресурси;

- збільшення вартості з ростом інфраструктури. Ліцензування базується на кількості датчиків, і для великих мереж ціна може швидко зрости;

- складність для новачків при масштабуванні. Для нових користувачів може бути складним налаштування розширеного моніторингу або управління великою кількістю датчиків [10].

Отже, Paessler PRTG є потужним і універсальним інструментом для моніторингу IT-інфраструктури, що підходить для організацій будь-якого масштабу. Він вирізняється легким налаштуванням, широкою підтримкою протоколів і ефективною системою сповіщень. Проте обмеження безкоштовної версії та значні вимоги до ресурсів можуть бути недоліками для великих організацій або тих, хто має обмежений бюджет. Загалом, це ефективне рішення для моніторингу, яке забезпечує високу якість контролю за IT-ресурсами.

Zabbix – це потужний та гнучкий інструмент для моніторингу IT-інфраструктури, призначений для забезпечення безперервного контролю за станом

серверів, мережевого обладнання, додатків та інших елементів інформаційної системи (рис. 1.4). Основною метою Zabbix є оперативне виявлення проблем, попередження про можливі збої та забезпечення надійного функціонування всіх критичних компонентів інфраструктури.

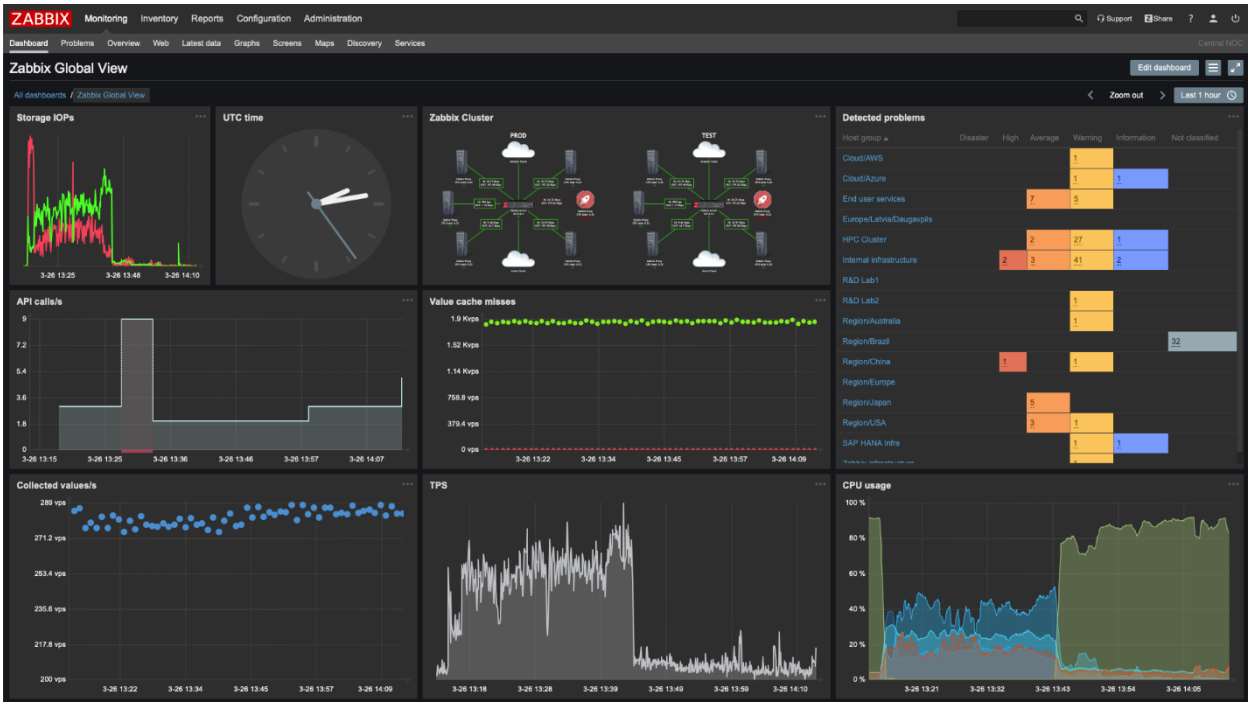


Рисунок 1.4 – Приклад інтерфейсу системи «Zabbix» [11]

Архітектура Zabbix побудована на основі клієнт-серверного підходу. Центральний компонент – це сервер Zabbix, який здійснює збір та обробку даних від моніторингових агентів, встановлених на віддалених пристроях, або через SNMP, IPMI, SSH та інші протоколи без встановлення агентів. Сервер зберігає всі дані у базі даних, що забезпечує доступність історичних даних для аналізу та звітності. Веб-інтерфейс Zabbix надає можливість користувачам налаштовувати та переглядати дані в режимі реального часу, а також працювати з графіками, звітами та налаштовувати оповіщення для критичних подій. Іншою важливою частиною архітектури є проксі Zabbix, який дозволяє масштабувати систему, розподіляючи навантаження між серверами, що є актуальним для великих мереж з багатьма віддаленими локаціями.

Переваги Zabbix:

- гнучкість моніторингу [12]. Підтримує різні методи збору даних (SNMP, IPMI, JMX, агентський та безагентський моніторинг), що дозволяє налаштовувати систему під конкретні потреби;

- масштабованість [13]. Підходить як для малих, так і для великих інфраструктур завдяки підтримці проксі-серверів, що розподіляють навантаження;

- реальний час. Миттєві сповіщення та моніторинг у реальному часі дозволяють швидко реагувати на критичні події;

- веб-інтерфейс. Інтуїтивний інтерфейс з налаштуванням дашбордів, звітів та графіків для зручного доступу до інформації.

Недоліки Zabbix:

- складність налаштування. Початкова конфігурація та адміністрування можуть потребувати значних зусиль, особливо для новачків;

- високі вимоги до ресурсів [14]. При роботі з великими інфраструктурами, Zabbix потребує значних апаратних ресурсів для забезпечення продуктивності;

- недостатня документація. Деякі складні аспекти роботи не завжди детально висвітлені в офіційній документації, що може ускладнювати освоєння;

- обмеження при безагентовому моніторингу. Менша гнучкість та можливості у порівнянні з агентським моніторингом, що може впливати на точність даних.

Отже, Zabbix є потужним і гнучким інструментом для моніторингу IT-інфраструктури, що дозволяє ефективно контролювати та керувати ресурсами як у невеликих, так і у великих мережах. Незважаючи на деякі недоліки, пов'язані зі складністю налаштування та високими вимогами до ресурсів, його масштабованість, підтримка багатьох протоколів та функціональність в реальному часі роблять його привабливим рішенням для організацій, що потребують надійного моніторингу.

1.2 Методи обробки та аналізу даних у системах моніторингу

У системах моніторингу інфраструктури для обробки та аналізу великих обсягів даних, зібраних у реальному часі, застосовуються різноманітні методи, які дозволяють своєчасно виявляти аномалії, прогнозувати можливі збої та забезпечувати стабільну роботу інфраструктурних систем. До основних методів, що використовуються для цього, належать методи статистичного аналізу, які допомагають оцінювати загальні показники та визначати відхилення від нормальних параметрів, і методи інтелектуального аналізу даних (data mining), що дозволяють виявляти приховані закономірності та аномалії у великих обсягах інформації. Завдяки поєднанню цих підходів системи моніторингу можуть своєчасно і точно реагувати на відхилення та потенційні загрози [15].

Статистичний аналіз використовується для оцінки загальних показників роботи системи, виявлення відхилень від нормальних параметрів та прогнозування можливих збоїв. Основними завданнями цих методів є ідентифікація аномальних ситуацій, аналіз трендів і поведінкових патернів системи, а також виявлення залежностей між різними параметрами інфраструктури.

У системах моніторингу статистичний аналіз застосовується для визначення базових показників, таких як середнє значення, медіана, мінімум і максимум, а також для оцінки варіативності даних за допомогою стандартного відхилення та дисперсії. Ці показники дозволяють встановити порогові значення, перевищення яких свідчить про потенційну проблему або збій у системі. Наприклад, якщо навантаження на процесор систематично перевищує допустимі межі, це може бути сигналом про критичне навантаження на систему або можливий збій.

На рис. 1.5 зображено різні методи статистичного аналізу, що застосовуються для моніторингу. Ці методи охоплюють аналіз середніх значень для оцінки загальної продуктивності системи, обчислення стандартного відхилення для виявлення ступеня варіативності даних, а також дисперсійний аналіз, що дозволяє оцінити, наскільки розкидані значення навколо середнього. На основі цих методів формується картина загальної стабільності системи та виявляються аномалії.

Рисунок 1.5 – Методи статистичного аналізу даних [16]

Окрім цього, важливу роль відіграє аналіз часових рядів, який дозволяє відстежувати зміни в роботі системи протягом певного періоду часу. Наприклад, зміни навантаження на мережеві пристрої або сервери можуть бути відображені як тренди, що допомагають передбачити майбутні проблеми. Аналіз часових рядів дозволяє виявляти довготривалі тенденції і реагувати на них, запобігаючи збоям на ранніх етапах.

Методи інтелектуального аналізу даних у системах моніторингу використовуються для виявлення прихованих закономірностей, трендів та аномалій у великих обсягах інформації. Ці методи дозволяють не лише збирати та обробляти дані, але й використовувати їх для побудови прогнозів, класифікації та ідентифікації потенційних загроз. В умовах динамічних змін і збільшення складності інфраструктурних систем методи інтелектуального аналізу даних допомагають знаходити ефективні рішення для підвищення надійності та продуктивності систем.

Основні завдання інтелектуального аналізу даних у системах моніторингу полягають у тому, щоб автоматично аналізувати великі масиви даних та на основі цього аналізу визначати закономірності, які можуть бути невидимими при використанні традиційних методів. До цих методів належать класифікація, кластеризація, асоціативні правила, регресійний аналіз, а також алгоритми виявлення аномалій.

На рис. 1.6 зображено основні методи інтелектуального аналізу даних, що використовуються для моніторингу інфраструктурних об’єктів. Класифікація є одним із ключових методів, який дозволяє розділяти дані на категорії або групи на основі певних ознак. Це корисно для виявлення патернів у роботі системи та визначення, до якої категорії належать ті чи інші події. Наприклад, класифікаційні алгоритми можуть використовуватися для визначення типу атаки на мережу або класу несправності обладнання.

Рисунок 1.6 – Методи інтелектуального аналізу даних [17]

Кластеризація дозволяє групувати події чи об’єкти, які мають схожі характеристики, без попередньо визначених міток. Це дозволяє системі виявляти нові аномалії або незвичні групи подій, які не підпадають під стандартну класифікацію. Такий підхід допомагає знаходити раніше невідомі або рідкісні збої в системі.

Методи асоціативних правил використовуються для виявлення закономірностей і залежностей між різними подіями в системі. Наприклад, збільшення навантаження на мережеві пристрої може бути пов’язане з конкретними часовими періодами або певними подіями, такими як надмірна активність користувачів. Виявлення таких залежностей дозволяє системі

моніторингу заздалегідь прогнозувати можливі проблеми та вживати запобіжних заходів.

Важливим методом є регресійний аналіз, який дозволяє визначати залежності між різними змінними та використовувати їх для прогнозування майбутніх подій. Це може бути корисно для прогнозування майбутнього стану системи на основі історичних даних. Наприклад, регресія дозволяє передбачити зростання навантаження на сервер або інші критичні ресурси системи.

Методи інтелектуального аналізу даних є незамінними інструментами для систем моніторингу, оскільки вони дозволяють виявляти приховані закономірності, визначати аномалії та здійснювати прогнозування, що допомагає підтримувати надійну та безперебійну роботу складних інфраструктурних об’єктів [18].

Таким чином, сучасні системи моніторингу інфраструктури використовують широкий спектр методів обробки та аналізу даних для забезпечення стабільної роботи систем, своєчасного виявлення аномалій та прогнозування можливих проблем. Інтеграція таких методів, як статистичний аналіз, інтелектуальний аналіз даних, виявлення аномалій та прогнозування збоїв, дозволяє підвищити ефективність моніторингу та зменшити ризики, пов’язані з несправностями та кіберзагрозами.

1.3 Принципи застосування машинного навчання у задачах моніторингу інфраструктури

Машинне навчання використовується в моніторингу інфраструктури завдяки своїй здатності обробляти великі обсяги даних і швидко реагувати на зміни в поведінці систем. Методи машинного навчання дозволяють системам ефективно працювати з різноманітними даними, автоматично виявляти проблеми та приймати рішення в реальному часі.

Однією з основних переваг машинного навчання є його здатність працювати з як структурованими, так і неструктурованими даними, що особливо важливо для

моніторингу інфраструктурних об’єктів, де джерела даних можуть бути різноманітними.

Аналіз структурованих даних за допомогою машинного навчання є досить ефективним завдяки тому, що ці дані вже мають чітку структуру (таблиці, бази даних) і можуть бути легко оброблені алгоритмами. Наприклад, у системах моніторингу серверів або мережі структуровані дані можуть включати такі показники, як завантаження процесорів, обсяг використаної пам’яті, мережевий трафік тощо [19].

Алгоритми машинного навчання, такі як дерева рішень та методи опорних векторів, забезпечують потужні засоби для класифікації структурованих даних. У моніторингу інфраструктури їх використовують для класифікації подій, визначення стану обладнання або об’єктів інфраструктури на основі попередньо визначених критеріїв. Класифікація допомагає виявити, чи перебуває певний елемент у нормальному стані або потребує негайного втручання.

Дерева рішень є одним із найпоширеніших та найзрозуміліших методів машинного навчання, які використовуються для вирішення задач класифікації та регресії. Цей метод ґрунтується на ієрархічній структурі, що дозволяє приймати рішення на основі послідовних запитань, відповіді на які визначають, до якого класу належить об’єкт або яке значення слід прогнозувати. Дерева рішень починаються з кореневого вузла, де задається початкове питання, що розбиває набір даних на дві або більше гілок залежно від відповіді. Кожна наступна гілка може вести до нових запитань або до кінцевого результату у листовому вузлі, який представляє кінцеве рішення для даного об’єкта.

Алгоритм побудови дерева рішень намагається максимізувати інформаційний приріст або зменшити ентропію на кожному кроці, щоб зробити дерево найбільш ефективним і точним. Приклад дерева рішення можна побачити на рис. 1.7, де зображено типову структуру такого дерева з кореневим вузлом, дочірніми або розділеними вузлами, та листовими вузлами, що відображають кінцеві результати класифікації або прогнозування.

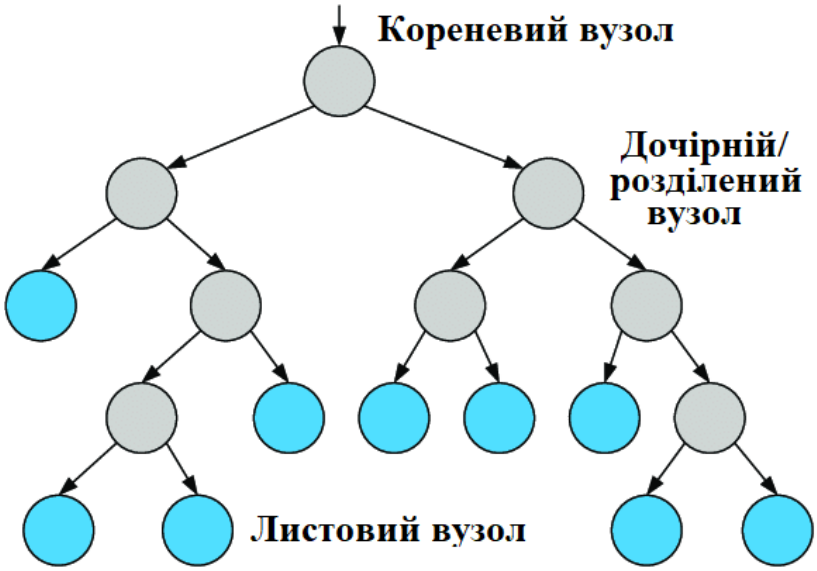


Рисунок 1.7 – Ілюстрація методу дерева рішень

Застосування машинного навчання у задачах моніторингу інфраструктури дозволяє системам автоматично аналізувати великі обсяги даних, що надходять з різних джерел, включаючи сенсори, камери спостереження та інші системи збору інформації. За допомогою алгоритмів, таких як дерева рішень, можна автоматично виявляти аномалії у роботі системи, прогнозувати можливі збої або несправності та здійснювати попереджувальні заходи для запобігання серйозних проблем.

Кластеризація дозволяє групувати схожі дані, наприклад, об’єкти інфраструктури, за певними параметрами. Алгоритми, такі як K-середніх або ієрархічна кластеризація, забезпечують автоматичне групування елементів без необхідності попереднього маркування, що полегшує аналіз великих структурованих наборів даних [20].

Ієрархічна кластеризація – це метод машинного навчання, який використовується для об’єднання подібних елементів у групи або кластери. Цей метод відрізняється від інших видів кластеризації тим, що він створює ієрархічну структуру, за допомогою якої кожен елемент або об’єкт даних спочатку розглядається як окремий кластер, а потім вони поступово об’єднуються у більші групи залежно від рівня схожості. Основна мета ієрархічної кластеризації полягає у тому, щоб створити дерево кластерів (дендрограму), яке відображає ієрархічні відносини між об’єктами. На кожному кроці алгоритм об’єднує два найближчих кластери, що дозволяє поступово формувати все більші угрупування.

На рис. 1.8 зображено ілюстрацію методу ієрархічної кластеризації, де можна побачити, як окремі об’єкти на перших кроках об’єднуються в пари, потім більші угрупування, і на завершальному етапі всі елементи поєднуються в один кластер. Кожен крок об’єднання показаний окремо, що дозволяє візуалізувати процес кластеризації та розуміти, як створюються кластери на кожному етапі.

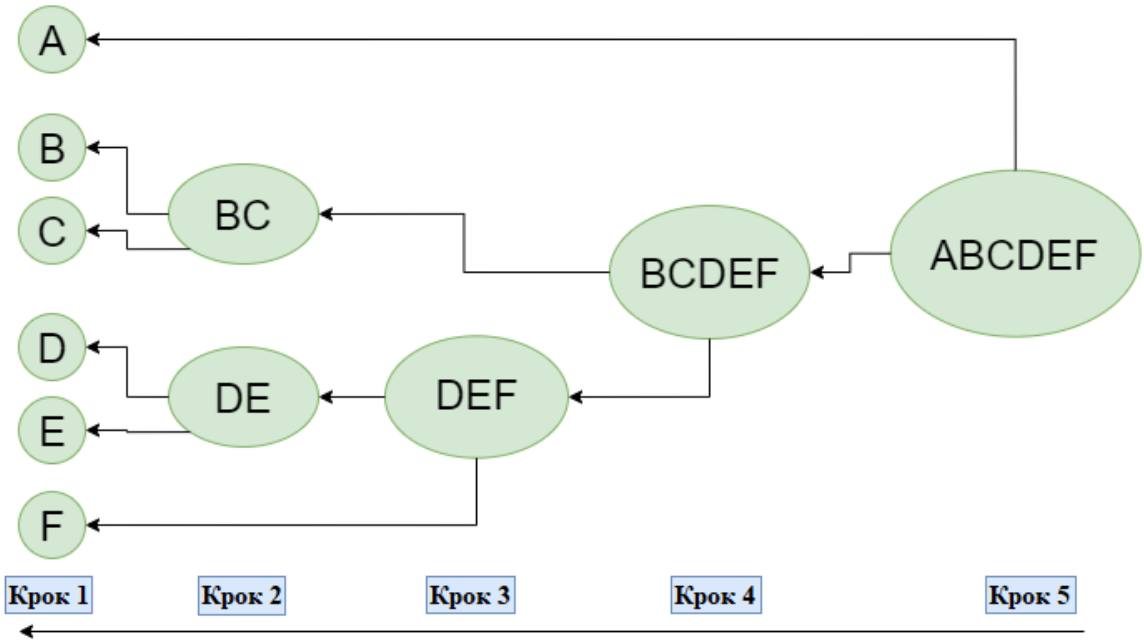


Рисунок 1.8 – Ілюстрація методу ієрархічної кластеризації

Також існує інший підхід, при якому процес починається з одного великого кластера, який потім поступово розділяється на менші кластери. Для вимірювання схожості між кластерами використовують різні метрики відстаней, такі як Евклідова або косинусна відстань.

Застосування методу ієрархічної кластеризації у задачах моніторингу інфраструктури має велике значення для аналізу та групування об’єктів, подій або інцидентів на основі їх характеристик. Наприклад, сенсори, що збирають інформацію про різні об’єкти інфраструктури, можуть передавати дані з великою кількістю параметрів, таких як температура, вологість, рівень зношеності, та інші фактори. Ієрархічна кластеризація дозволяє об’єднувати ці дані в логічні групи на основі схожих характеристик, що допомагає операторам швидко ідентифікувати подібні елементи, визначати закономірності та виявляти потенційні проблеми або несправності.

Цей метод також корисний для визначення типів поведінки або відхилень в роботі системи, які можуть вказувати на можливі аварії або несправності. Наприклад, якщо декілька сенсорів починають показувати схожі відхилення від нормальних показників, ієрархічна кластеризація дозволяє об’єднати ці інциденти в один кластер і швидко визначити можливі причини. Це значно спрощує процес прийняття рішень щодо ремонту або обслуговування інфраструктурних об’єктів, підвищуючи ефективність їх моніторингу та управління [21].

Застосування машинного навчання для аналізу неструктурованих даних дозволяє ефективно працювати з інформацією, що надходить у різних форматах, таких як тексти, зображення, аудіо або відео файли. Вони дають змогу витягувати корисну інформацію з цих різнорідних джерел, роблячи процес аналізу даних більш гнучким та адаптивним до умов і типів даних.

Одним із ключових завдань при роботі з неструктурованими даними є аналіз тексту. Алгоритми обробки природної мови (Natural Language Processing, NLP) дозволяють машинному навчанню ідентифікувати важливі сутності у великих текстових масивах, такі як назви об’єктів, дати, події та інші критично важливі відомості. Це забезпечує автоматизований моніторинг новинних стрічок, технічної документації або звітів про стан інфраструктури, що може бути корисним для прийняття швидких рішень на основі аналізу цих даних.

Метод обробки природної мови дозволяє комп’ютерам аналізувати, розуміти та генерувати текст, подібний до того, як це робить людина. NLP включає в себе різні методи та техніки для аналізу текстових даних, такі як токенізація, морфологічний аналіз, розпізнавання сутностей та аналіз емоцій. Цей метод застосовується для роботи з текстами у будь-якій природній мові та дозволяє витягати корисну інформацію з текстових даних, таких як новини, документи, повідомлення користувачів, що надходять у неструктурованому вигляді [22].

Алгоритми NLP працюють за допомогою аналізу лінгвістичних особливостей тексту та його структури, що дозволяє системам автоматично витягати сутності, проводити семантичний аналіз, розуміти контекст тексту, а також виявляти емоції або ставлення до подій. Це дозволяє зменшити

навантаження на людей при роботі з великими масивами текстових даних, забезпечуючи автоматизований процес обробки та аналізу.

На рис. 1.9 зображено ілюстрацію процесу обробки природної мови, де показано, як текстові дані, отримані з різних джерел, таких як мобільні пристрої або соціальні мережі, надходять у шар NLP для подальшої обробки. Цей шар виконує аналіз тексту, після чого результати можуть бути збережені в базі знань або у сховищі даних для подальшої аналітики. Така схема дозволяє інтегрувати дані з різних джерел, об'єднуючи їх в одну систему для аналітичного опрацювання.

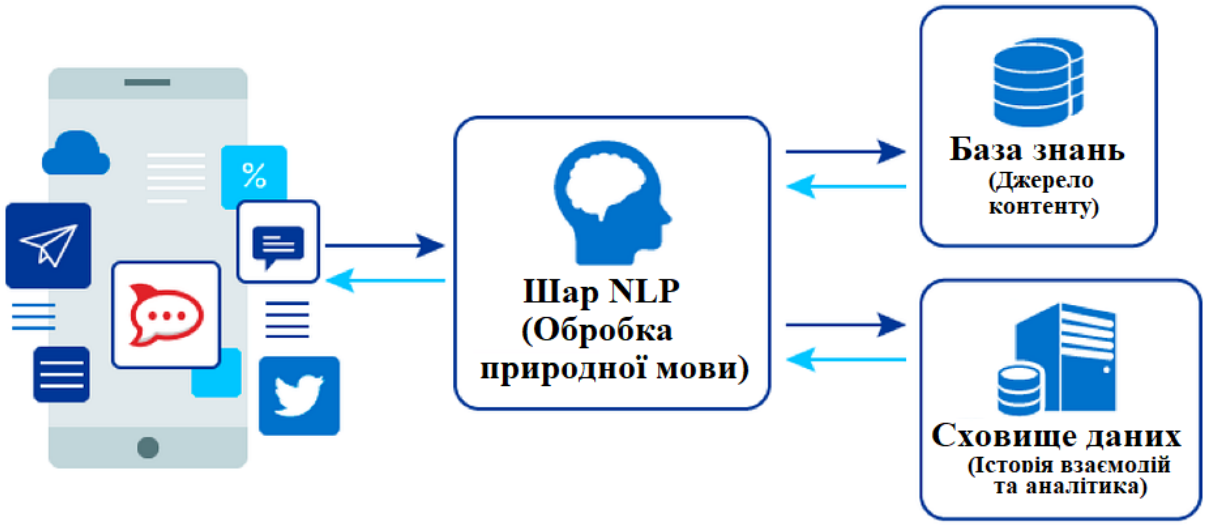


Рисунок 1.9 – Ілюстрація методу обробки природної мови

У задачах моніторингу інфраструктури метод NLP застосовується для роботи з неструктурованими даними, що можуть надходити у вигляді текстових звітів, повідомлень або інших видів документів. Наприклад, в умовах міської інфраструктури різні повідомлення про несправності, отримані через соціальні мережі або спеціалізовані платформи, можуть бути автоматично проаналізовані за допомогою NLP для виявлення проблемних зон або категорій несправностей. Це дозволяє ефективніше працювати з великими обсягами текстових даних, які надходять у неструктурованій формі, та автоматизувати процес обробки інформації, полегшуючи операторам прийняття рішень на основі цих даних [23].

Метод глибокого навчання, зокрема згорткові нейронні мережі (CNN), використовується для обробки складних типів даних, таких як зображення, відео або інші неструктуровані дані. CNN працюють за принципом автоматичного

виділення ознак з вхідних даних. Це досягається за допомогою згорткових шарів, які застосовують фільтри для витягу ключових особливостей зображень, таких як контури, текстури або інші важливі елементи. Ці особливості передаються через кілька шарів мережі, що дозволяє моделі поступово будувати уявлення про структуру даних на вищому рівні абстракції.

CNN складається з кількох згорткових і пулінгових шарів. Згорткові шари виконують операцію згортки, проходячи по вхідному зображенню і виділяючи важливі ознаки, в той час як пулінгові шари зменшують розмірність даних, зберігаючи найважливіші характеристики. Після кількох шарів згортки і пулінгу результуючі ознаки "вирівнюються" і передаються до повнозв’язаного шару для остаточної класифікації.

На рис. 1.10 зображено схему роботи згорткової нейронної мережі. Вхідне зображення проходить через кілька згорткових шарів, де відбувається виділення ознак, а потім через шари пулінгу, які спрощують дані, зберігаючи ключові характеристики. На останньому етапі вирівняні ознаки передаються до повнозв’язаного шару, де відбувається процес класифікації, що дозволяє системі прийняти рішення, до якого класу належить даний об’єкт.

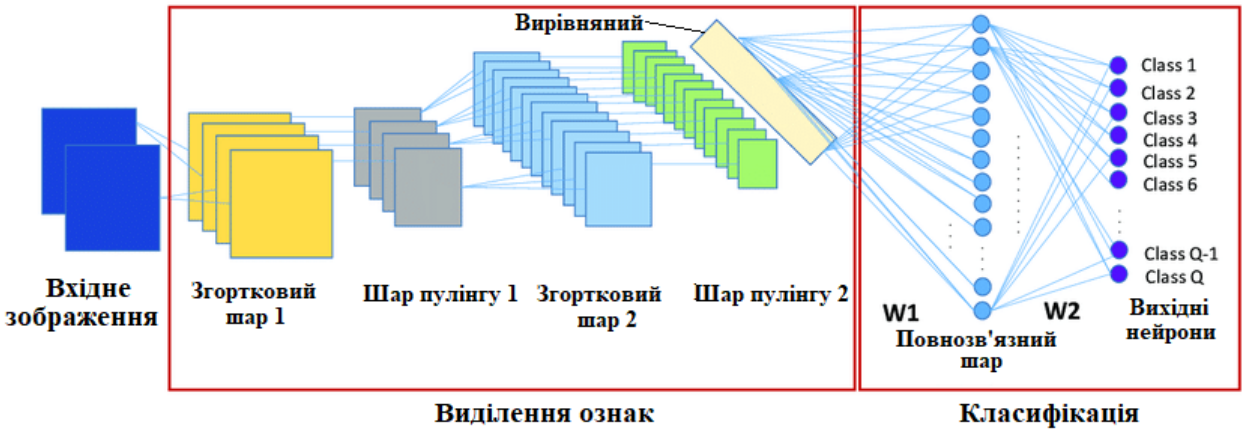


Рисунок 1.10 – Ілюстрація методу глибокого навчання

У задачах моніторингу інфраструктури згорткові нейронні мережі використовуються для обробки та аналізу неструктурованих даних, таких як відеоспостереження або зображення інфраструктурних об’єктів. Наприклад, система моніторингу, яка використовує камери для спостереження за мостами, дорогами або будівлями, може застосовувати CNN для автоматичного виявлення

пошкоджень або інших аномалій на основі зображень. Мережа здатна виділяти ключові ознаки, такі як тріщини, зношені елементи або деформації, і класифікувати їх за рівнем небезпеки. Це дозволяє значно полегшити роботу операторів, адже система самостійно виявляє потенційні проблеми та повідомляє про них [24]. Таким чином, CNN дозволяє ефективно працювати з великими обсягами неструктурованих даних у вигляді зображень або відео, забезпечуючи точний і автоматизований аналіз.

1.4 Методи класифікації та їх роль у моніторингу: огляд найбільш ефективних підходів

Класифікація є одним із ключових методів машинного навчання, який знаходить широке застосування в задачах моніторингу інфраструктури. Основна мета методів класифікації полягає у розподілі даних на певні категорії або класи, що дозволяє системам моніторингу автоматично визначати тип подій або проблем та відповідним чином реагувати на них. Використання класифікаційних методів значно підвищує точність виявлення аномалій, дозволяє прогнозувати майбутні збої та забезпечує більш оперативне реагування на загрози. У цьому підрозділі розглянуто найбільш ефективні методи класифікації, їх принципи роботи та роль у забезпеченні стабільної роботи інфраструктурних об’єктів.

Методи класифікації працюють на основі навчання моделей на великому обсязі історичних даних, які вже були відмічені або позначені певними класами (напр., нормальні події або збої). Після навчання модель може автоматично класифікувати нові дані, які не мають попередньо визначених міток. У системах моніторингу це означає, що нові події або сигнали можуть автоматично відноситися до певних категорій, що допомагає визначити їхній тип і прийняти відповідні заходи для усунення проблем або запобігання збоям.

Класифікація є особливо важливою у випадках, коли потрібно швидко і точно ідентифікувати тип проблеми або загрози. Вона дозволяє системам моніторингу не лише виявляти відхилення від норми, але й автоматично розуміти їхню природу, що допомагає вживати належні заходи у режимі реального часу.

У моніторингу інфраструктури використовуються різні методи класифікації, кожен з яких має свої переваги залежно від характеру даних та вимог до системи. Нижче наведено огляд найбільш ефективних методів класифікації, що широко застосовуються у задачах моніторингу [25].

Метод опорних векторів

Метод опорних векторів (Support Vector Machine, SVM) є одним із ефективних інструментів для класифікації, який використовується у багатьох сферах, включаючи моніторинг інфраструктури. На рис. 1.11 продемонстровано структуру SVM, яка включає вхідний шар, прихований шар із застосуванням ядра, та вихідний шар. Вхідні дані передаються до прихованого шару, де відбувається обчислення ядрових функцій $K(X, X_i)$, що дозволяє перетворювати початкові дані у простір вищої розмірності для кращого розділення класів. На виході відбувається сумування з вагами та зміщенням, яке забезпечує розмежування між класами, що й дозволяє проводити ефективну класифікацію.

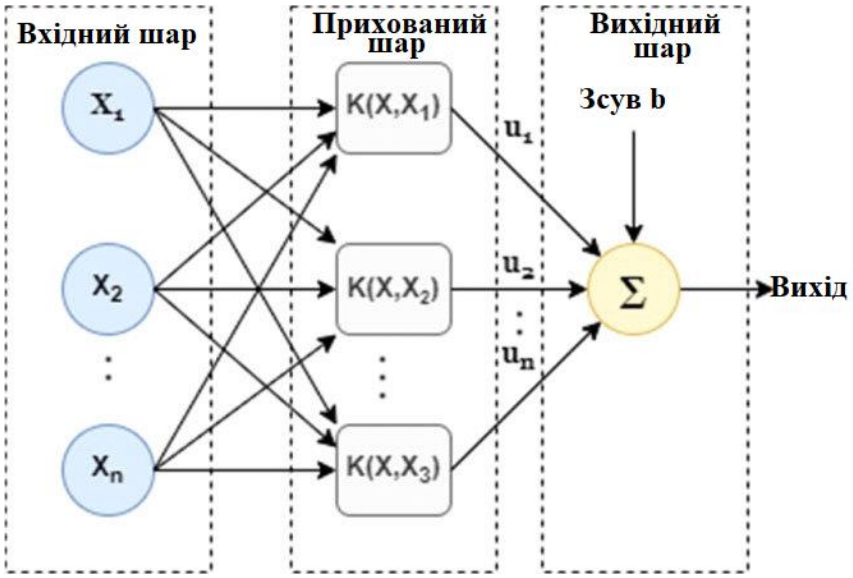


Рисунок 1.11 – Схема роботи методу SVM

Перевага SVM полягає в його здатності працювати з великими наборами даних та добре розрізняти складні класи, навіть якщо вони не є лінійно віддільними. Наприклад, для виявлення кіберзагроз SVM може розділяти трафік на нормальний та підозрілий, що дозволяє виявляти атаки у режимі реального часу [26].

Випадковий ліс

Метод випадкового лісу є одним із ефективних підходів до класифікації, який використовує ансамбль дерев рішень для прийняття остаточного рішення. Ця модель ґрунтується на створенні кількох дерев рішень, де кожне дерево навчається на випадковій вибірці даних і приймає окреме рішення. Після цього результати об'єднуються за допомогою механізму, відомого як бегіг (голосування більшості), де кожне дерево "голосує" за свій результат, а клас, за який проголосувала більшість дерев, стає остаточним результатом класифікації.

На рис. 1.12 представлено процес навчання і тестування моделі на основі методу випадкового лісу. На етапі навчання створюються кілька дерев рішень, кожне з яких робить своє передбачення щодо належності об'єкта до класу А або класу В. Після цього, на етапі тестування, результати всіх дерев об’єднуються через механізм голосування більшості, що і дає кінцевий прогноз щодо класу об'єкта. Це дозволяє зменшити вплив одного дерева, яке могло би зробити помилковий висновок, оскільки загальний результат залежить від консенсусу серед дерев.

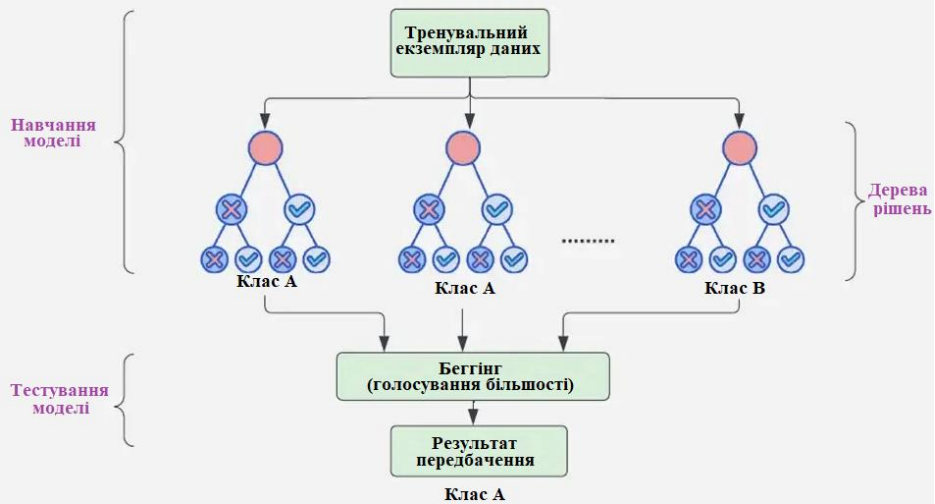


Рисунок 1.12 – Схема роботи методу випадкового лісу

У задачах моніторингу інфраструктури метод випадкового лісу застосовується для класифікації та аналізу великих обсягів даних, що надходять з різних джерел, таких як сенсори або мережеві системи. Він дозволяє виявляти аномалії у роботі обладнання або систем, аналізуючи безліч параметрів одночасно. Наприклад, у системах моніторингу серверів даний метод може допомагати у класифікації типів збоїв, аналізуючи характеристики навантаження на сервери, час відгуку та інші параметри. У промислових системах цей метод може

використовуватися для виявлення дефектів на ранніх стадіях, аналізуючи показники, такі як температура, тиск, рівень вібрації або споживання енергії, що дозволяє запобігти серйозним поломкам та зупинкам виробництва [27].

Наївний байєсівський класифікатор

Наївний байєсівський класифікатор (НБК) є одним із базових методів класифікації, що працює на основі теореми Байєса. Основна ідея методу полягає в оцінці ймовірності належності об'єкта до певного класу на основі ймовірностей для кожної ознаки об'єкта за припущення про незалежність цих ознак. На рис. 1.13 продемонстровано, як цей алгоритм приймає вхідні дані з різними ознаками (форми та кольори) і класифікує їх за трьома різними групами. Алгоритм розраховує ймовірність приналежності кожного об'єкта до одного з класів на основі формули Байєса: ймовірність $P(A|B)$ розраховується через ймовірності $P(B|A)$, $P(A)$ та $P(B)$.

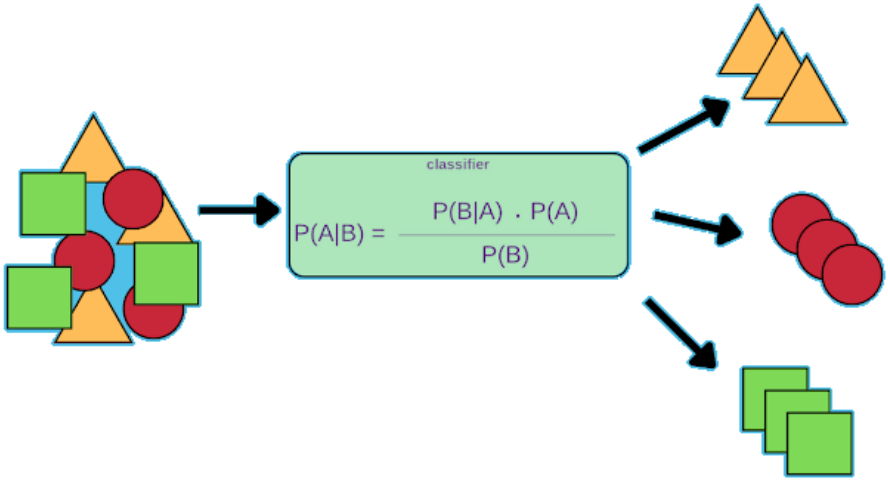


Рисунок 1.13 – Схема роботи методу НБК

Даний метод, хоча і базується на припущенні про незалежність ознак, показує досить високу ефективність у багатьох реальних задачах, завдяки своїй швидкості і здатності працювати з великими наборами даних. Враховуючи швидкість обробки, метод НБК добре підходить для роботи у реальному часі.

У задачах моніторингу інфраструктури наївний байєсівський класифікатор може бути застосований для класифікації подій у мережах, швидкого аналізу великих обсягів даних та виявлення аномальних ситуацій. Наприклад, в системах моніторингу мережевих трафіків або сенсорних даних наївний байєс може швидко класифікувати потоки даних як нормальні або підозрілі. Це дозволяє системам

моніторингу оперативно реагувати на потенційні загрози або несправності, забезпечуючи швидку діагностику можливих проблем. Метод також може використовуватися для виявлення фішингових атак або кіберзагроз, що дозволяє зменшити ризики збоїв у роботі інфраструктури або мережевих систем [28].

Отже, методи класифікації є важливими інструментами для ефективного моніторингу інфраструктури. Вони забезпечують автоматизацію процесів аналізу та прийняття рішень, дозволяють виявляти потенційні проблеми на ранніх стадіях та підвищують стабільність роботи систем. Класифікація є незамінною у випадках, коли необхідно не лише виявити проблему, але й точно визначити її тип та природу, що допомагає оптимізувати реагування та забезпечити надійну роботу інфраструктури.

1.5 Використання штучного інтелекту для виявлення атак на інфраструктурні об’єкти

Сучасні системи моніторингу інфраструктурних об'єктів стають все більш складними, оскільки кількість і характер загроз продовжують еволюціонувати. З цієї причини використання штучного інтелекту (ШІ) для виявлення та попередження атак на такі об'єкти є критично важливим. Системи на основі штучного інтелекту можуть обробляти величезні обсяги даних, що надходять від різних джерел, включаючи мережеву активність, сенсорні дані з обладнання, активність користувачів, додатків та баз даних. Ці дані, будучи неструктурованими, потребують ефективних інструментів аналізу для виявлення потенційних загроз у режимі реального часу.

Методи штучного інтелекту, такі як машинне навчання, дозволяють системам аналізувати історичні дані, навчатися на прикладах минулих атак та прогнозувати нові загрози на основі поточних даних. Алгоритми глибокого навчання здатні виявляти складні патерни та аномалії, які можуть вказувати на атаку, навіть якщо такі патерни раніше не були чітко вираженими або документованими.

Основний принцип роботи таких систем полягає в тому, що ШІ моделі проходять кілька етапів – від навчання на історичних даних до тестування на нових наборах даних. Після цього модель починає працювати у продуктивному середовищі, аналізуючи поточні дані для виявлення ознак атаки. Однією з головних переваг таких систем є їх здатність працювати в режимі реального часу та інтегруватися з існуючими інфраструктурами, надаючи візуалізовану інформацію для оперативного прийняття рішень [29].

На рис. 1.14 зображено загальну схему виявлення атак за допомогою штучного інтелекту.

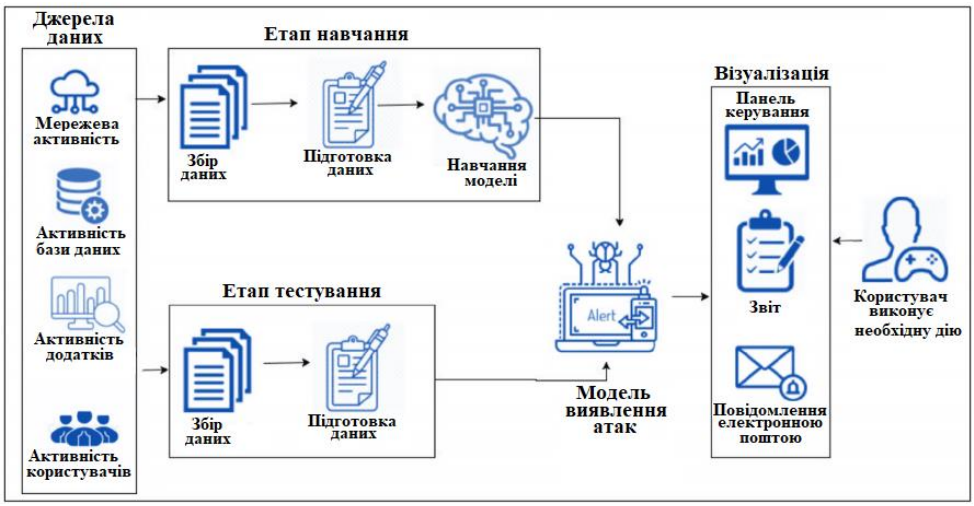


Рисунок 1.14 – Загальна схема виявлення атак за допомогою ШІ

На першому етапі здійснюється збір даних з різних джерел, таких як мережева активність, активність баз даних, додатків і користувачів. Далі ці дані готуються для навчання моделі, що включає попередню обробку та підготовку відповідних наборів характеристик. Після етапу навчання модель тестується на нових даних, що надходять із тих же джерел.Модель виявлення атак аналізує нові дані і на основі своїх прогнозів генерує відповідні попередження. Ці попередження можуть бути відображені на панелі керування, надіслані у вигляді звітів або повідомлень електронною поштою для оперативного реагування. Така інтеграція ШІ у системи моніторингу дозволяє забезпечити ефективне управління безпекою інфраструктурних об'єктів та знизити ризики критичних атак.

Одним із найбільш поширених прикладів використання ШІ є виявлення розподілених атак типу DDoS (Distributed Denial of Service). ШІ-системи

аналізують мережевий трафік у реальному часі і виявляють підозрілі зміни у його інтенсивності та патернах. Використання класифікаційних алгоритмів дозволяє ідентифікувати аномалії у трафіку, які вказують на атаку, що дозволяє своєчасно реагувати на загрозу та мінімізувати її наслідки.

В енергетичних системах ШІ використовується для моніторингу дій у мережах розподілу електроенергії, виявлення спроб несанкціонованого доступу або втручання у роботу обладнання. Алгоритми глибокого навчання аналізують велику кількість сенсорних даних та виявляють аномальні дії, що можуть свідчити про спробу втручання у систему керування енергомережами.

ШІ також застосовується у системах водопостачання для виявлення можливих кібератак, спрямованих на керування насосами, клапанами та іншими компонентами мережі. Наприклад, атаки на SCADA-системи можуть бути виявлені за допомогою алгоритмів машинного навчання, що аналізують дані з сенсорів та виявляють аномальні команди, які можуть призвести до аварій або збоїв у системі [30].

Використання штучного інтелекту для виявлення атак на інфраструктурні об’єкти стало важливим компонентом сучасних систем кібербезпеки. Завдяки методам машинного навчання та глибокого навчання, системи ШІ здатні не лише виявляти відомі типи атак, але й прогнозувати нові загрози, які раніше були непоміченими.

2 ВИБІР ТЕХНОЛОГІЧНОГО СТЕКУ ТА ОПИС МЕТОДОЛОГІЇ РОЗРОБКИ

2.1 Вибір технологічного стеку для розробки системи моніторингу інфраструктури

Для створення ефективної та надійної системи моніторингу інфраструктури важливо вибрати відповідний технологічний стек, зокрема мову програмування, яка забезпечить необхідну функціональність, продуктивність та масштабованість. Вибір мови впливає на здатність системи обробляти великі обсяги даних у реальному часі, інтегруватися з іншими компонентами та зручно масштабуватися під потреби користувача. Серед популярних мов програмування, які підходять для розробки таких систем, варто виділити Python, Java та C#.

Python – це високорівнева мова програмування, яка забезпечує простоту написання коду та має велику кількість бібліотек для роботи з даними та обчисленнями [31]. Завдяки своїй гнучкості та підтримці численних фреймворків, Python часто використовується для аналізу даних та розробки серверної частини систем. Мова є чудовим вибором для побудови прототипів, а також для сценаріїв, що потребують високої інтерактивності з користувачем.

Java є однією з найпопулярніших мов для створення масштабованих систем, завдяки своїй переносимості між платформами та надійності [32]. Вона має розвинену екосистему та підтримує багатопотоковість, що дозволяє обробляти велику кількість одночасних процесів. Крім того, Java ідеально підходить для побудови мережевих та розподілених додатків, що є важливим аспектом у системах моніторингу.

C# – потужна мова програмування, яка широко використовується у розробці корпоративних додатків, особливо в екосистемі Microsoft [33]. Вона забезпечує високу продуктивність та інтеграцію з іншими продуктами, що робить її відмінним вибором для систем, які потребують стабільної роботи та масштабованості. Завдяки підтримці новітніх технологій, C# також зручна для розробки складних серверних додатків.

Для ґрунтовного аналізу доцільності використання кожної мови програмування у розробці системи моніторингу інфраструктури варто порівняти їх основні характеристики. Це дозволяє об'єктивно оцінити переваги та недоліки кожної мови на основі якісних і кількісних показників, зокрема продуктивності, можливостей масштабування та підтримки сучасних інструментів. У табл. 2.1 наведено ключові характеристики мов Python, Java та C# для подальшого вибору оптимального рішення.

Таблиця 2.1 – Порівняння основних характеристик мов програмування

Характеристика	Python	Java	C#
Продуктивність (виконання коду)	Середня	Висока	Дуже висока
Підтримка багатопотоковості	Обмежена	Висока	Висока
Переносимість	Висока	Висока	Висока (в межах .NET екосистеми)
Інтеграція з корпоративними системами	Обмежена	Висока	Дуже висока
Простота навчання	Висока	Середня	Середня
Підтримка новітніх технологій	Висока (особливо у сфері AI)	Висока	Дуже висока
Час компіляції	Немає (інтерпретована)	Довгий	Короткий
Спільнота розробників	Дуже велика	Велика	Велика

Мова програмування C# обрана для розробки системи моніторингу інфраструктури завдяки її високій продуктивності та оптимізованому часу виконання коду, що особливо важливо для систем реального часу. Підтримка багатопотоковості у C# забезпечує ефективну обробку паралельних процесів, що критично для великих інфраструктурних рішень. Крім того, C# вирізняється високим рівнем інтеграції з корпоративними системами, зокрема в екосистемі Microsoft, що спрощує інтеграцію з іншими бізнес-інструментами. Завдяки

активній підтримці новітніх технологій, C# дозволяє швидко адаптувати систему до сучасних вимог та

Для побудови ефективної системи моніторингу інфраструктури необхідний надійний вибір системи управління базами даних (СУБД), яка забезпечить стабільне зберігання, швидкий доступ до інформації та підтримку обробки великих обсягів даних у реальному часі. Правильний вибір СУБД впливає на продуктивність, масштабованість і можливості інтеграції системи, зокрема в умовах великих підприємств та динамічного розширення інфраструктури. Серед найпоширеніших варіантів СУБД, які розглядаються для таких цілей, можна виділити MS SQL Server, MySQL та FireBird.

MS SQL Server – СУБД від Microsoft, яка забезпечує високу продуктивність, стабільність та інтеграцію з іншими продуктами Microsoft [34]. Вона має розвинені можливості для аналізу даних, підтримує складні запити, а також оптимізована для багатокористувацького середовища. Завдяки потужним інструментам безпеки, MS SQL Server є популярним вибором для корпоративних рішень.

MySQL – відкрита та широко використовувана СУБД, яка відзначається швидкістю та простотою налаштування [35]. Вона ідеально підходить для веб-додатків і підтримує широке коло платформ, що робить її гнучкою для використання в різних проєктах. MySQL забезпечує високу надійність і є одним із найбільш економічних рішень для зберігання великих обсягів даних.

FireBird – вільна СУБД з відкритим кодом, яка підтримує можливості транзакцій та відзначається низьким споживанням ресурсів [36]. Ця система забезпечує високу продуктивність навіть при обмежених ресурсах, що робить її хорошим вибором для компактних систем. FireBird має підтримку SQL-стандартів і підходить як для малих, так і для середніх додатків.

Для більш детального порівняння особливостей кожної СУБД слід звернути увагу на основні характеристики, які визначають їх продуктивність, гнучкість і зручність використання в умовах різних інфраструктурних вимог. Такий аналіз дозволить оцінити переваги кожної системи управління базами даних з огляду на потреби проєкту, що включає високі вимоги до стабільності та швидкості обробки

даних. У табл. 2.2 наведено основні характеристики MS SQL Server, MySQL та FireBird для обґрунтування вибору оптимальної СУБД.

Таблиця 2.2 – Порівняння основних характеристик СУБД

Характеристика	MS SQL Server	MySQL	FireBird
Підтримка транзакцій	ACID-повна	ACID-повна	ACID-повна
Масштабованість	Висока	Середня	Середня
Розвинені засоби аналітики	Розширені інструменти BI	Обмежені	Обмежені
Інтеграція з іншими системами	Глибока, особливо з продуктами Microsoft	Висока для веб-додатків	Обмежена
Підтримка великих обсягів даних	Висока	Висока	Обмежена
Засоби для забезпечення безпеки	Розширені механізми (ролі, права доступу, шифрування)	Базові можливості безпеки	Базові можливості безпеки
Підтримка багатокористувацького середовища	Розширена	Середня	Обмежена

Виходячи з аналізу в табл. 2.2, MS SQL Server є оптимальним вибором для розробки системи моніторингу інфраструктури завдяки високій масштабованості, що дозволяє ефективно працювати з великими обсягами даних. Розвинені засоби аналітики та інтеграція з іншими продуктами Microsoft забезпечують зручність для побудови бізнес-інтелекту та інтеграційних рішень. MS SQL Server також пропонує розширені механізми безпеки, що є важливим для захисту критичних даних системи. Підтримка багатокористувацького середовища робить його придатним для масштабних проєктів, де потрібна стабільна робота з численними одночасними запитами.

2.2 Вибір інструментів і технологій для обробки даних та машинного навчання

Моніторинг інфраструктури з використанням методів машинного навчання потребує потужних інструментів для обробки та аналізу великих обсягів даних у реальному часі. Сучасні фреймворки та платформи, такі як ML.NET, TensorFlow і Keras, надають необхідні ресурси та бібліотеки для реалізації складних моделей машинного навчання, оптимізованих для великих даних і розподілених обчислень. Ці інструменти забезпечують багатофункціональні модулі для швидкого створення, навчання та оцінки моделей, що дозволяє ефективно обробляти інфраструктурні дані й отримувати точні прогнози для підтримки стабільності та безпеки системи.

ML.NET – це потужна платформа від Microsoft для створення моделей машинного навчання, оптимізована для інтеграції з .NET екосистемою. Вона забезпечує інструменти для побудови, навчання та розгортання моделей без необхідності використовувати сторонні сервіси, що є важливою перевагою для розробників .NET додатків [37]. ML.NET підтримує широкий спектр завдань машинного навчання, таких як класифікація, регресія, кластеризація та інші, дозволяючи легко інтегрувати ці моделі в існуючі .NET проєкти. Завдяки відкритій архітектурі, платформа підтримує не лише класичні алгоритми, але й глибоке навчання через інтеграцію з ONNX і TensorFlow. Це робить її універсальною для роботи з різними типами даних і дозволяє адаптувати систему для специфічних завдань.

На рис. 2.1 зображена архітектура, яка демонструє процес обробки вхідних зображень і генерацію карт політності. Ця архітектура включає мережу для вилучення ознак, яка отримує вхідне зображення та генерує карту фіксації людини. Далі результати проходять через функцію втрат і використовуються для навчання моделі.

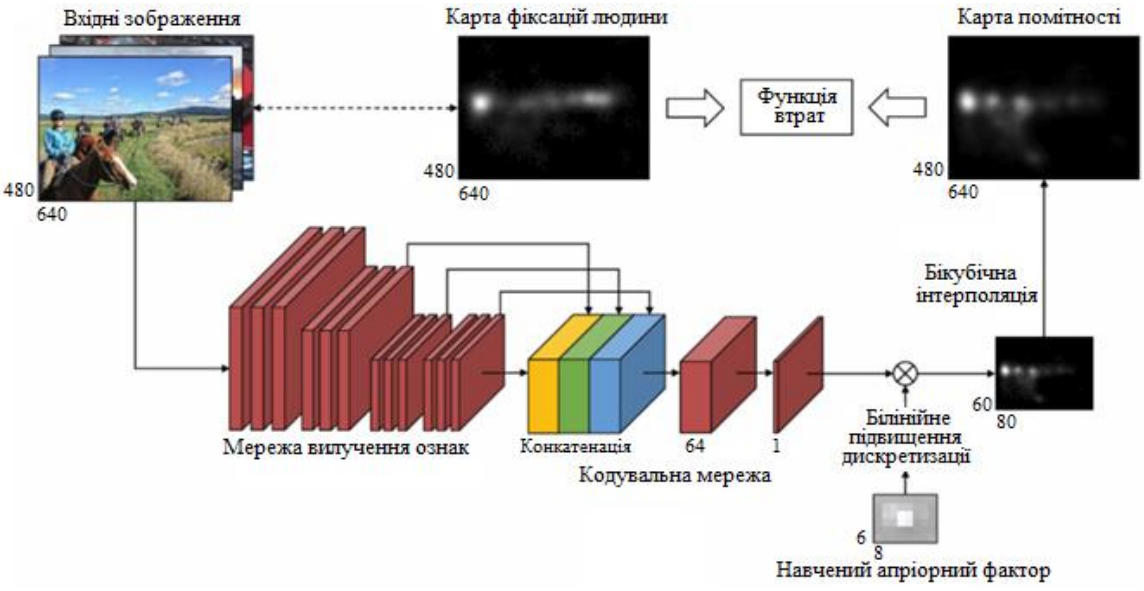


Рисунок 2.1 – Архітектура ML.NET [38]

Кодувальна мережа та модулі конкатенації відповідають за об'єднання вилучених ознак, забезпечуючи високу точність розпізнавання. Завдяки інтерполяції та підвищенню дисперсії ця архітектура дозволяє створювати точні карти політності, що сприяє покращенню продуктивності та якості моделі.

Переваги архітектури:

- висока точність розпізнавання [39]. Завдяки поєднанню мережі вилучення ознак і кодувальної мережі досягається висока точність результатів;

- гнучкість адаптації. Архітектура дозволяє змінювати параметри на різних етапах, що забезпечує налаштування під специфічні завдання;

- можливість інтерполяції. Використання кубічної інтерполяції підвищує деталізацію вихідних даних, що особливо важливо для точних прогнозів;

- апріорний фактор навчання. Наявність цього фактору дозволяє налаштовувати модель на основі попередніх даних, що покращує її стабільність.

Недоліки архітектури:

- високі обчислювальні ресурси. Для роботи з великими зображеннями потрібні значні апаратні ресурси, що може обмежувати її застосування;

- складність налаштування. Кількість параметрів і етапів налаштування роблять архітектуру складною для розробників;

- часове навантаження. Через багатоступеневий процес обробки час виконання може бути значним, що не завжди прийнятно для реальних задач у реальному часі;

- залежність від якості вхідних даних. Низька якість вхідних зображень може призвести до втрати точності та стабільності моделі.

Отже, дана архітектура є потужним інструментом для обробки зображень та побудови карт політності, що забезпечує високу точність і гнучкість у налаштуванні. Однак, через вимогливість до обчислювальних ресурсів та складність налаштування, вона може бути недоцільною для систем з обмеженими ресурсами або в задачах, що потребують швидкої обробки в режимі реального часу.

TensorFlow – це одна з найпопулярніших платформ для машинного навчання та глибокого навчання, розроблена Google [40]. Вона надає інструменти для побудови, тренування та розгортання моделей нейронних мереж з підтримкою як для дослідницьких, так і для виробничих середовищ. TensorFlow пропонує широкий спектр API, що дозволяє використовувати його як для базових обчислень, так і для створення складних багатошарових нейронних мереж. Окрім роботи на процесорах (CPU), TensorFlow підтримує графічні процесори (GPU) і спеціалізовані тензорні процесори (TPU), що значно прискорює навчання моделей на великих наборах даних. Це робить платформу універсальною та дозволяє її застосовувати у різних областях, включаючи комп'ютерний зір, обробку природної мови, рекомендаційні системи тощо.

Рис. 2.2 відображає архітектуру TensorFlow, яка демонструє процес підготовки даних, розподілу обчислень та варіанти розгортання моделей. Ця архітектура включає попередню обробку даних, яка виконується через tf.data і feature columns для забезпечення стандартизації та підготовки даних перед навчанням.

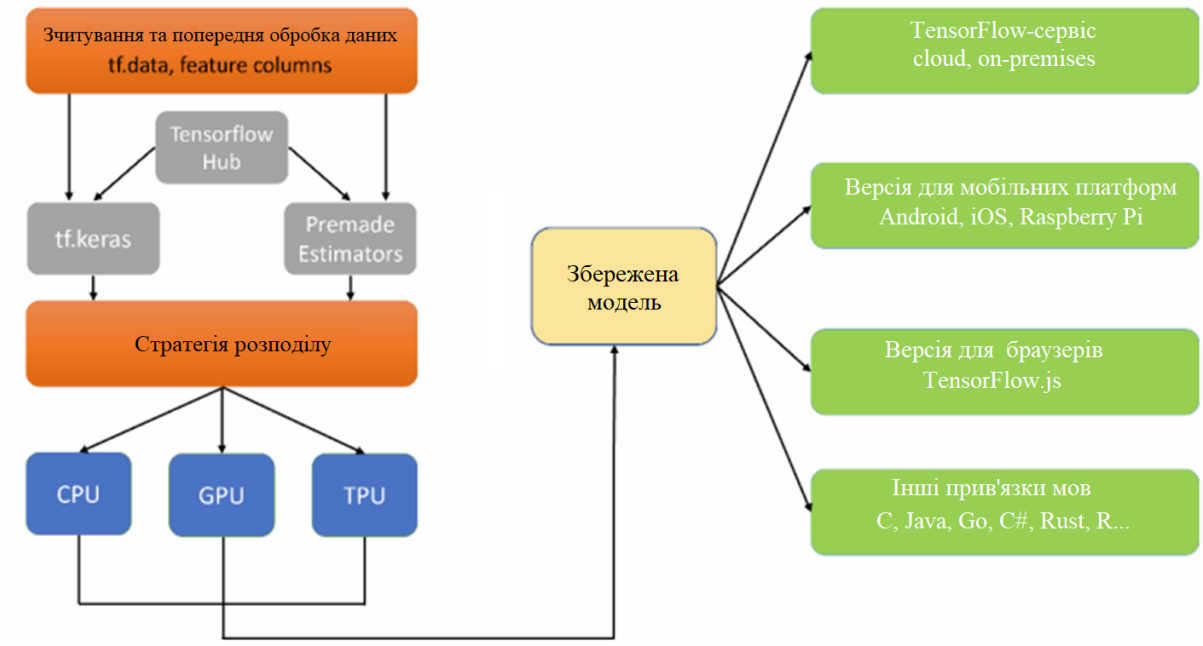


Рисунок 2.2 – Архітектура TensorFlow [41]

Модель може бути розподілена на різні обчислювальні платформи, такі як CPU, GPU та TPU, що забезпечує гнучкість у виборі ресурсів залежно від складності задачі. Після збереження модель може бути розгорнута на різних платформах: у хмарних сервісах TensorFlow, на мобільних пристроях, у веб-браузерах за допомогою TensorFlow.js, а також інтегрована в інші мовні середовища, такі як C, Java, Go та C#.

Переваги архітектури TensorFlow:

- універсальність обробки даних [42]. Платформа забезпечує потужні інструменти для попередньої обробки даних, що спрощує підготовку наборів даних до навчання моделей;

- гнучкий розподіл ресурсів. Підтримка CPU, GPU та TPU дозволяє ефективно розподіляти обчислювальні навантаження відповідно до потреб проєкту, що забезпечує масштабованість;

- мультиплатформене розгортання. Моделі можна розгорнути на різних платформах, включаючи хмарні сервіси, мобільні пристрої та веб-браузери, що розширює можливості використання;

- інтеграція з багатьма мовами програмування. TensorFlow підтримує інтеграцію з мовами C, Java, Go, C# тощо, що дозволяє використовувати його в різних розробницьких середовищах.

Недоліки архітектури:

- складність освоєння. Високий поріг входу та необхідність знання різних компонентів платформи можуть ускладнити роботу для новачків;

- великі вимоги до ресурсів. Для роботи з великими моделями на GPU та TPU потрібні значні апаратні ресурси, що може бути недоступним для малих проєктів;

- відсутність оптимізації для специфічних завдань. TensorFlow не завжди може надати оптимальні рішення для вузькоспеціалізованих задач, що може потребувати додаткових налаштувань;

- обмеження в мобільному розгортанні. На мобільних пристроях продуктивність може бути обмеженою, що впливає на роботу ресурсомістких моделей.

Отже, архітектура TensorFlow є надзвичайно потужною та гнучкою для різних завдань машинного навчання, забезпечуючи широкий спектр можливостей для обробки даних, розподілу обчислень і мультиплатформеного розгортання. Водночас вона може вимагати значних ресурсів та спеціалізованих знань, що робить її оптимальним вибором для великих та середніх проєктів, але менш підходящою для малих систем із обмеженими ресурсами або специфічними вимогами.

Keras – це високорівнева бібліотека для машинного навчання, яка надає простий інтерфейс для побудови та тренування нейронних мереж. Вона розроблена для швидкої розробки прототипів та інтеграції складних моделей глибокого навчання з мінімальними витратами часу на конфігурацію. Завдяки підтримці таких фреймворків, як TensorFlow та Theano, Keras дозволяє розробникам реалізовувати моделі для різних завдань, включаючи класифікацію, обробку

зображень та тексту [43]. Особливістю Keras є її інтуїтивний API, що спрощує процес навчання моделей і підвищує ефективність розробки. Ця бібліотека широко використовується як у дослідницьких проєктах, так і у виробничих системах, забезпечуючи гнучкість та потужність у налаштуванні різних типів шарів і архітектур нейронних мереж.

На рис. 2.3 зображена архітектура нейронної мережі, створеної за допомогою Keras, що демонструє процес класифікації вхідного зображення. Ця архітектура складається з вхідного шару, кількох згорткових шарів, шарів пулінгу, повністю під'єднаних шарів та шару Softmax.

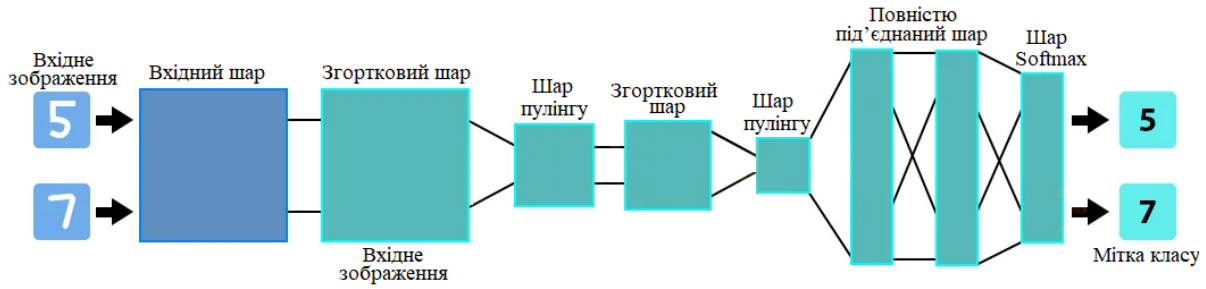


Рисунок 2.3 – Архітектура Keras [44]

Вхідне зображення проходить через згорткові шари, де відбувається вилучення ознак, а шари пулінгу зменшують розмірність для підвищення ефективності. Після цього дані потрапляють у повністю під'єднані шари, які обробляють ознаки та передають їх на шар Softmax для остаточної класифікації, визначаючи ймовірність приналежності до кожного класу.

Переваги архітектури Keras:

- ефективне вилучення ознак. Завдяки згортковим шарам відбувається якісне вилучення важливих ознак із зображення, що підвищує точність класифікації;

- зменшення розмірності [45]. Шари пулінгу дозволяють скоротити обсяг даних, зменшуючи обчислювальні витрати та покращуючи швидкість роботи моделі;

- гнучкість у налаштуванні. Наявність повністю під’єднаних шарів дає змогу адаптувати модель до різних типів завдань класифікації;

- чітка імовірнісна оцінка. Шар Softmax забезпечує ймовірнісний розподіл виходів, що дозволяє точно визначити клас об'єкта.

Недоліки архітектури:

- високі обчислювальні витрати. Глибокі згорткові шари можуть вимагати значних апаратних ресурсів, що ускладнює використання на пристроях з обмеженими ресурсами;

- чутливість до якості вхідних даних. Знижена якість зображень або шуми можуть негативно вплинути на точність класифікації;

- ризик перенавчання. Через велику кількість параметрів існує ймовірність перенавчання, особливо при недостатньому обсязі навчальних даних;

- обмежена інтерпретованість. Складність архітектури може ускладнити інтерпретацію результатів та розуміння, як модель приймає рішення.

Отже, архітектура забезпечує високу ефективність у задачах класифікації зображень завдяки потужному вилученню ознак і зменшенню розмірності, що оптимізує обробку даних. Проте високі обчислювальні вимоги та ризик перенавчання обмежують її застосування у деяких випадках, особливо для задач з обмеженими ресурсами. Архітектура підходить для складних задач, але вимагає належного обсягу даних та ресурсів для забезпечення стабільної роботи й високої точності.

Для детального аналізу доцільності використання кожної з бібліотек машинного навчання варто порівняти їх за основними характеристиками, що відображають продуктивність, зручність інтеграції та можливості розгортання. Це дозволить об’єктивно оцінити, наскільки кожен з інструментів відповідає вимогам до системи моніторингу інфраструктури. У табл. 2.3 наведено порівняння ключових характеристик ML.NET, TensorFlow та Keras для вибору оптимального рішення.

Таблиця 2.3 – Порівняння основних характеристик бібліотек

Характеристика	ML.NET	TensorFlow	Keras
Інтеграція з .NET	Повна	Обмежена	Обмежена
Простота використання для C#	Висока	Середня	Середня
Оптимізація для Windows	Висока	Середня	Низька
Локальне розгортання	Швидке та ефективне	Залежить від підтримки обладнання	Залежить від TensorFlow
Підтримка моделей AutoML	Повна	Часткова	Відсутня
Підтримка класичних алгоритмів	Висока	Середня	Низька

Згідно з аналізом у таблиці 2.3, ML.NET є оптимальним вибором для розробки системи моніторингу інфраструктури завдяки повній інтеграції з екосистемою .NET, що спрощує розробку для середовища C#. Висока оптимізація для Windows дозволяє ефективно використовувати цю бібліотеку на серверах, що працюють у корпоративних інфраструктурах, побудованих на цій операційній системі. Окрім того, підтримка AutoML в ML.NET сприяє автоматизації вибору і налаштування моделей, що зменшує потребу у вузькоспеціалізованих знаннях у машинному навчанні. Завдяки цьому ML.NET забезпечує ефективне локальне розгортання і підтримку класичних алгоритмів, що робить її привабливим рішенням для швидкої і надійної розробки системи моніторингу.

2.3 Методологія для розробки системи штучного інтелекту для моніторингу інфраструктури

Основна відмінність методології розробки інформаційного забезпечення для систем моніторингу інфраструктури на основі штучного інтелекту від традиційних підходів полягає в тому, що етапи аналізу та синтезу взаємопов'язані та можуть повторюватися кілька разів залежно від результатів попередніх обчислень. У цьому підході інтеграція аналізу та синтезу під час розробки дозволяє динамічно адаптувати систему до змін у даних і умовах функціонування, що є важливим аспектом для системи, яка потребує навчання та самовдосконалення.

Основні етапи аналізу в процесі проектування інформаційного забезпечення для системи штучного інтелекту включають кілька ключових кроків, спрямованих на забезпечення ефективного моніторингу та виявлення аномалій у функціонуванні інфраструктури, зокрема:

- створення математичної моделі для опису інфраструктури та її станів: цей етап полягає у формалізації основних компонентів інфраструктури та їх взаємодій, що дозволяє системі відстежувати нормальні стани та зміни в них. Математична модель служить основою для розуміння нормальних і відхилених станів, що є критично важливим для подальшого виявлення аномалій і прогнозування можливих збоїв;

- моделювання процесів виявлення аномалій: оскільки аномалії в інфраструктурі можуть бути складними і різноманітними, створення моделей для їх виявлення вимагає використання адаптивних підходів. Цей етап включає розробку алгоритмів для виявлення різних типів аномалій, враховуючи їх можливу випадковість та непередбачуваність;

- дослідження вхідних даних для визначення типових патернів і аномалій: на цьому етапі аналізуються великі обсяги історичних даних, щоб ідентифікувати закономірності у поведінці системи. Це дозволяє системі визначити характерні патерни нормальної роботи інфраструктури, а також виділити ознаки, що можуть свідчити про потенційні збої або проблеми. Дослідження даних є важливим етапом, оскільки воно дає змогу виділити ключові індикатори, на основі яких алгоритми ШІ будуть навчатися розпізнавати аномальні стани.

На етапі формування вхідного математичного опису вирішуються такі задачі:

- формування набору ознак для моніторингу;

- забезпечення репрезентативності навчальної вибірки для підвищення точності моделі;

- визначення контрольних значень для ключових метрик, що дозволяє встановити діапазони допустимих коливань для кожного показника інфраструктури.

На рис. 2.4 зображено ієрархічну структуру задач аналізу, які виникають при проектуванні інформаційного забезпечення для інтелектуальних систем моніторингу інфраструктури.

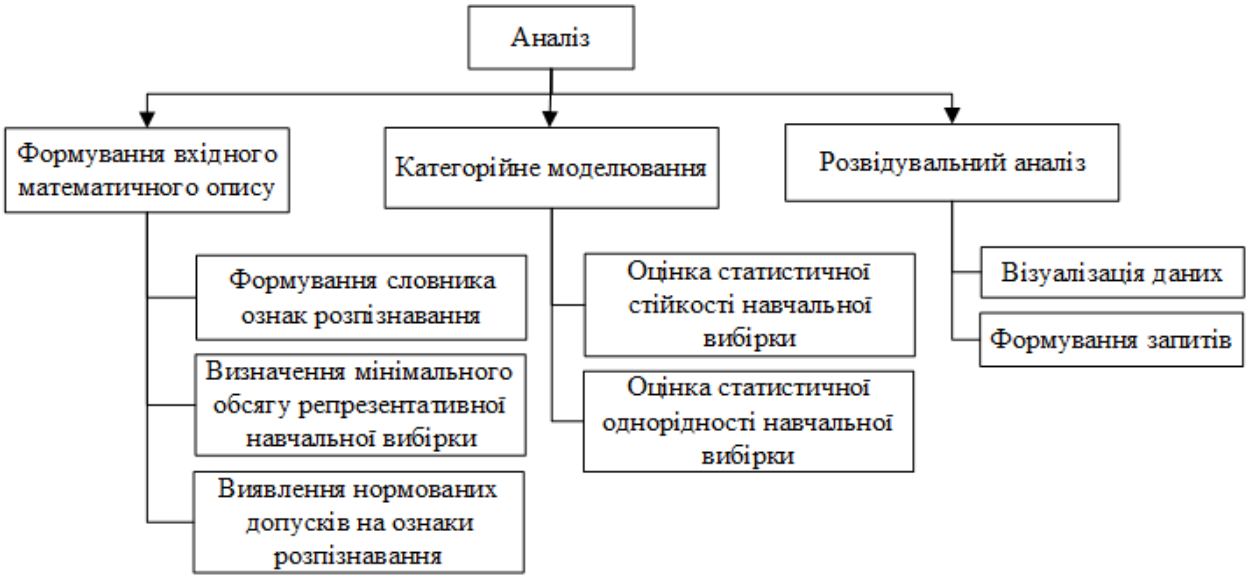


Рисунок 2.4 – Ієрархічна структура задач аналізу

Основними задачами початкового аналізу є:

- оцінка стабільності та однорідності вибірки для створення навчальної матриці, що використовується у кластерному аналізі, а також для виявлення закономірностей у даних з метою уточнення алгоритмів навчання системи;

- візуалізація даних для коригування процесу навчання, а також для вивчення тенденцій підвищення точності, швидкості та надійності роботи системи;

- оцінка релевантності даних, що дозволяє виявити корисні, аномальні, заважаючі (шумові) та приховані ознаки, які можуть вплинути на ефективність роботи алгоритмів моніторингу.

Після визначення основних етапів аналізу та підготовки даних для проектування інформаційного забезпечення системи штучного інтелекту, важливо перейти до побудови ефективного алгоритму навчання моделі. Цей алгоритм

покликаний забезпечити точність і надійність системи у виявленні аномалій у інфраструктурі. Процес навчання передбачає підготовку конвеєра обробки даних, нормалізацію, побудову моделі та її оцінку. Кожен із цих кроків є критично важливим для створення надійної системи, здатної розпізнавати аномалії на основі аналізу різних характеристик і патернів даних.

На рис. 2.5 представлений алгоритм навчання моделі для виявлення аномалій, що показує послідовність кроків, необхідних для успішного навчання моделі.

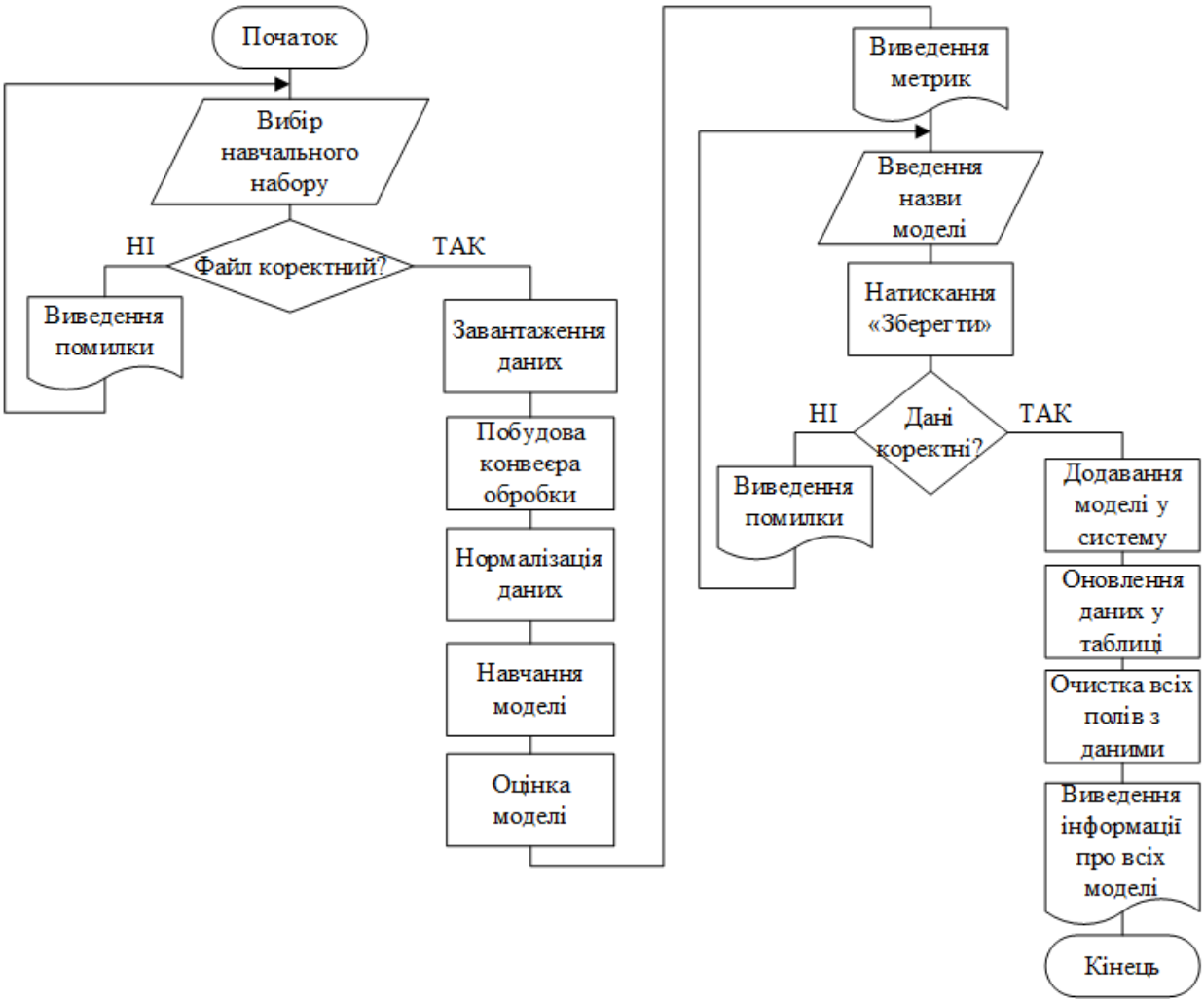


Рисунок 2.5 – Алгоритм навчання моделі для виявлення аномалій

На початку обирається навчальний набір даних, який проходить перевірку на коректність. Якщо файл має помилки, система виводить відповідне повідомлення, і необхідно усунути ці проблеми перед продовженням. Далі, за умови коректності файлу, виконується завантаження даних, після чого побудова конвеєра обробки готує дані для навчання. На етапі нормалізації дані приводяться до

стандартизованого вигляду, що дозволяє моделі ефективно працювати з різнорідними показниками. Після цього запускається навчання моделі, яка вивчає характерні патерни, що можуть вказувати на аномалії. На завершальному етапі проводиться оцінка моделі для визначення її точності та ефективності, а у випадку необхідності – коригування та повторне навчання.

Після навчання моделі для виявлення аномалій наступним кроком є інтеграція її у систему для проведення безперервного моніторингу мережевого трафіку. На рис. 2.6 зображений алгоритм моніторингу мережевого трафіку, який показує основні етапи роботи системи після запуску.

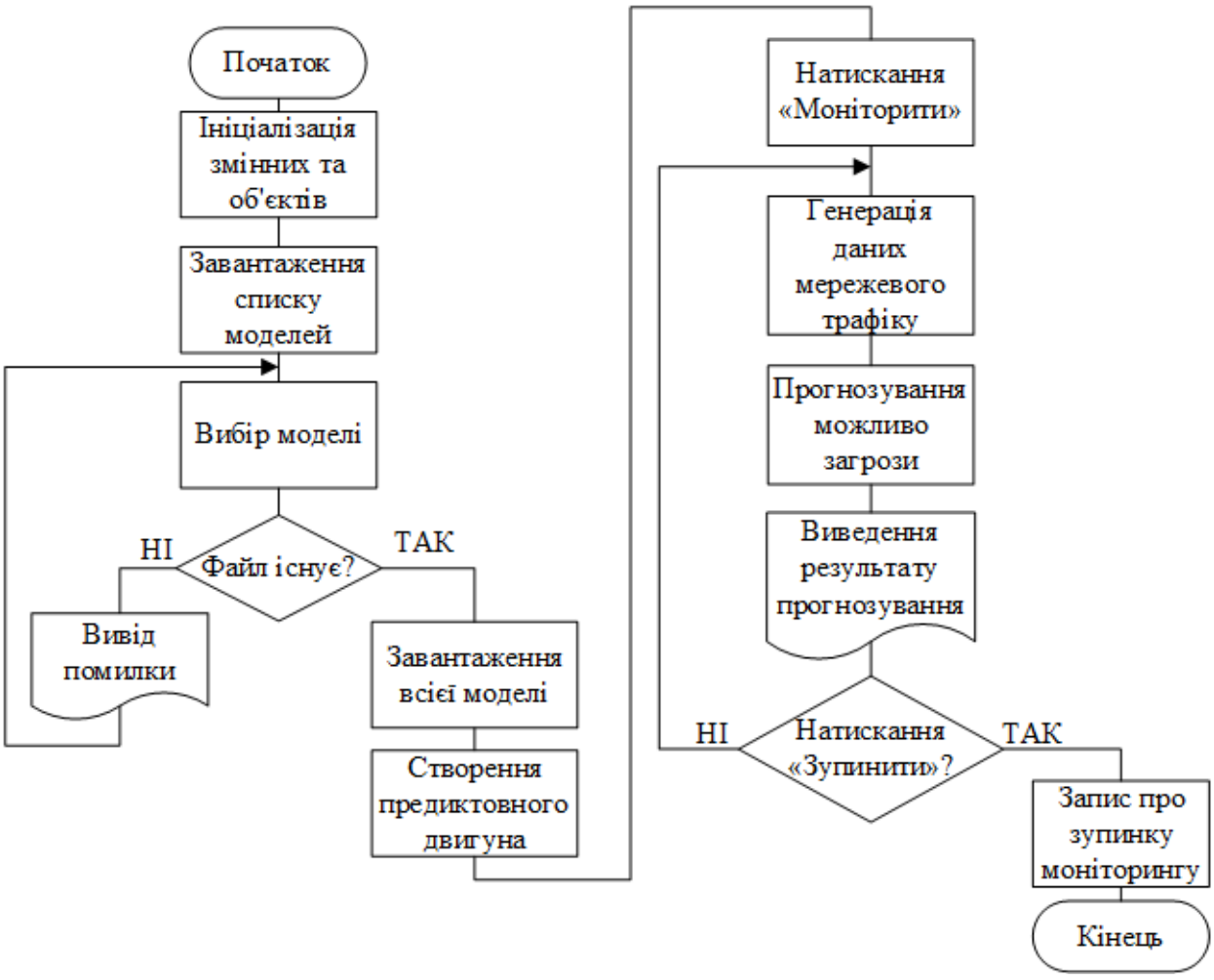


Рисунок 2.6 – Алгоритм моніторингу мережевого трафіку

На початку відбувається ініціалізація змінних та об’єктів, що включає завантаження списку моделей та вибір тієї, яка буде використовуватися для моніторингу. Після перевірки існування файлу з моделлю система завантажує необхідні компоненти, створює предиктивний двигун і переходить до стадії

активного моніторингу, натисканням кнопки «Моніторити». На цьому етапі система починає генерувати дані мережевого трафіку, аналізуючи їх для прогнозування можливих загроз. Якщо загроза виявлена, результат прогнозування виводиться користувачеві. Процес триває доти, доки не натиснуто кнопку «Зупинити», після чого система записує інформацію про зупинку моніторингу та завершує роботу.

3 РОЗРОБКА СИСТЕМИ ВИЯВЛЕННЯ АТАК НА ІНФРАСТРУКТУРУ ЗА ДОПОМОГОЮ ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Постановка задачі виявлення атак на основі багатокласової класифікації

Інфраструктурні системи, такі як мережеві середовища, енергетичні мережі та обчислювальні комплекси, генерують величезні обсяги телеметричних даних. Ці дані необхідно ефективно обробляти для виявлення аномалій, прогнозування несправностей та забезпечення стабільності роботи компонентів. Однак сучасні підходи, які базуються на ручному аналізі даних або статичних правилах, часто виявляються неефективними в умовах динамічного навантаження та складності взаємодії між компонентами.

Постає необхідність розробки інтелектуальної системи, яка б автоматизувала процес аналізу даних, підвищила швидкість реакції на інциденти та забезпечила адаптивність до змін у роботі інфраструктури. Така система має використовувати алгоритми штучного інтелекту для багатофакторного аналізу телеметричних даних, виявлення аномалій у реальному часі та прогнозування можливих несправностей.

1. Проблематика та ключові виклики, які потребують вирішення:

- ручна обробка подій і даних. Адміністратори витрачають значний час на аналіз великих обсягів телеметрії, що збільшує ризик пропущених інцидентів і затримок у прийнятті рішень;

- недостатня точність традиційних методів. Використання статичних порогів для виявлення аномалій не враховує динамічний характер інфраструктури та може спричиняти хибнопозитивні або хибнонегативні результати;

- обмежені можливості масштабування. Зі зростанням інфраструктури та обсягів даних традиційні системи аналізу стають непрактичними;

- неможливість прогнозування. Більшість сучасних рішень орієнтовані на реактивний підхід, коли інциденти обробляються лише після їхнього виникнення.

2. Завдання розробки та основні етапи реалізації:

- розробка підсистеми збору даних та забезпечення попередньої обробки даних для усунення шуму, нормалізації та підготовки до аналізу;

- реалізація модуля аналізу даних. Використання алгоритму машинного навчання для класифікації аномалій та прогнозування можливих несправностей. Забезпечення можливості виявлення нетипових змін у роботі інфраструктури;

- розробка модуля моніторингу. Побудова графічного інтерфейсу для візуалізації поточного стану системи. Інтеграція зі сповіщеннями для оперативного повідомлення про виявлені аномалії;

- інтеграція модуля аналітики . Формування звітів щодо продуктивності компонентів інфраструктури та аналізу інцидентів. Надання засобів для аналізу тенденцій і оптимізації роботи системи.

3. Очікувані результати:

- автоматизація процесів моніторингу. Система дозволить зменшити навантаження на адміністраторів, замінивши ручний аналіз телеметрії автоматизованими алгоритмами;

- підвищення точності виявлення аномалій. Використання моделей ШІ знизить кількість позитивних та хибних результатів;

- масштабованість. Система буде адаптована до збільшення обсягів даних та кількості моніторингових об’єктів.

Реалізація цієї системи сприятиме підвищенню надійності та ефективності роботи інфраструктури, дозволить оптимізувати управління ресурсами та знизити ризики, пов’язані з інцидентами.

3.2 Проектування системи моніторингу інфраструктури з використанням штучного інтелекту

Для проєктування системи моніторингу інфраструктури з використанням штучного інтелекту було обрано трирівневу архітектуру через її здатність

забезпечувати масштабованість, модульність та адаптивність. Розподіл функцій між рівнями дозволяє ізолювати обробку даних, логіку застосунку та інтерфейс користувача, що спрощує розробку і підтримку системи. Такий підхід забезпечує ефективну обробку великих обсягів телеметричних даних, можливість інтеграції нових алгоритмів машинного навчання та стійкість до змін в інфраструктурі.

Рівень даних забезпечує збір, зберігання та передачу інформації між іншими компонентами системи. Діаграма цього рівня наведена на рис. 3.1.

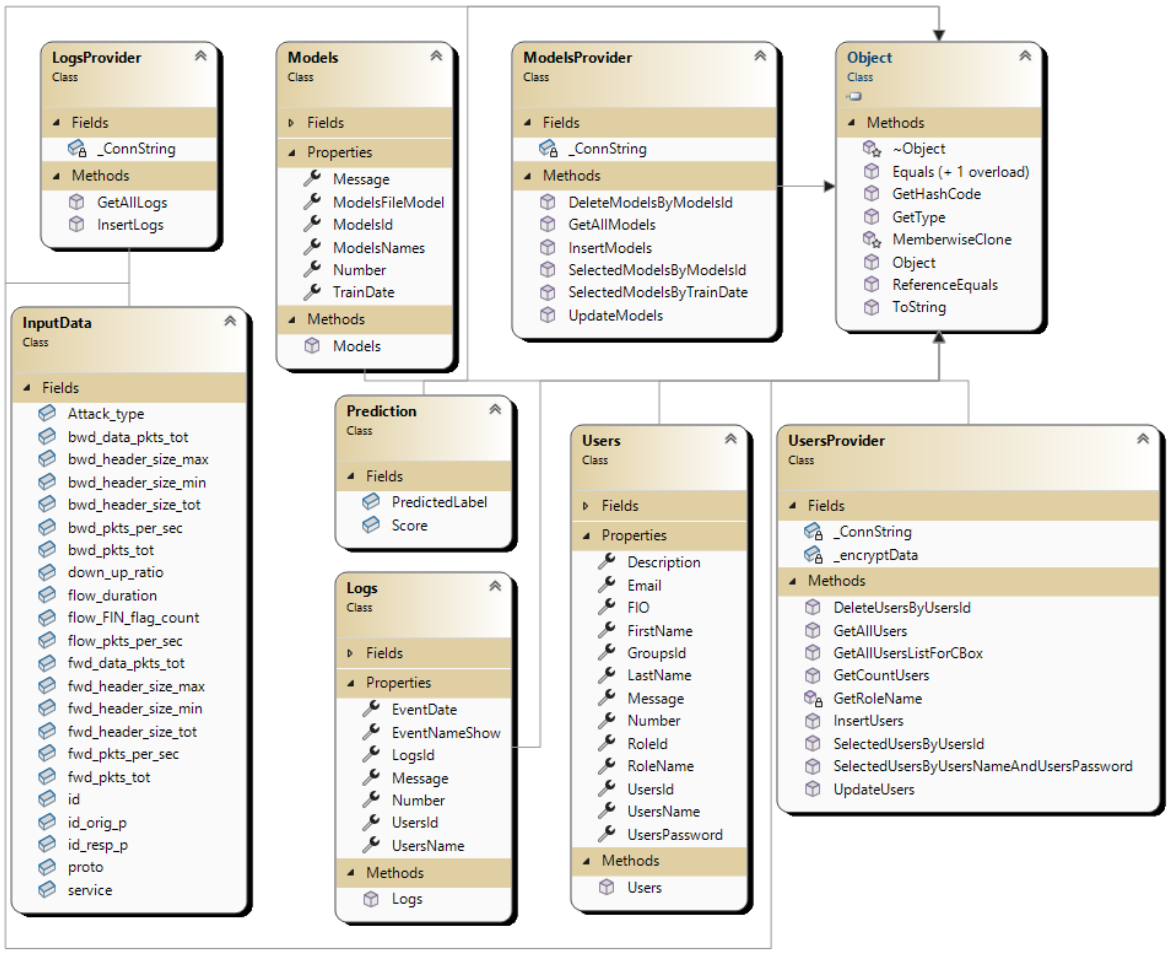


Рисунок 3.1 – Діаграма класів рівня даних

Основу діаграми становлять наступні класи:

- Users – цей клас призначений для зберігання даних про користувачів системи. Він містить такі основні атрибути, як ім'я користувача, унікальний ідентифікатор, роль у системі, електронна пошта та інші дані. Цей клас слугує контейнером для передачі інформації про користувачів між різними компонентами системи;

- UsersProvider – клас, який реалізує логіку роботи з користувачами. Він містить методи для отримання, додавання, оновлення та видалення інформації про користувачів, а також забезпечує доступ до даних через шифрування та використання підключення до бази даних;

- Models – клас, що містить дані про моделі машинного навчання, які використовуються для аналізу інфраструктури. Основними атрибутами є назва моделі, її тип, дата створення та інші дані, необхідні для її ідентифікації й використання;

- ModelsProvider – забезпечує функції управління моделями, включаючи додавання, видалення та оновлення інформації про моделі в базі даних. Цей клас дозволяє отримувати списки моделей та працювати з ними у процесах аналізу даних;

- Logs – клас для зберігання інформації про події в системі. Основними властивостями є дата, тип події, опис і дані користувача, який ініціював подію. Клас призначений для фіксації подій з метою моніторингу та аналізу роботи системи;

- LogsProvider – реалізує методи для управління даними журналів, включаючи збереження нових записів та отримання історії подій із бази даних;

- InputData – клас для зберігання та передачі телеметричних даних, які аналізуються в системі. Він містить широкий набір атрибутів, таких як параметри мережевого трафіку, ідентифікатори та характеристики потоку;

- Prediction – клас для представлення результатів роботи моделей машинного навчання. Містить передбачену модельну мітку класу та оцінки ймовірностей для кожного класу.

Така структура рівня даних забезпечує чіткий розподіл обов'язків між класами, полегшуючи їхнє використання та підтримку, а також підвищуючи ефективність роботи системи загалом.

Рівень інтерфейсу системи забезпечує взаємодію користувача з функціоналом системи через графічні елементи, які дозволяють виконувати

основні операції: авторизацію, управління користувачами, роботу з моделями машинного навчання та моніторинг процесів. Діаграма класів рівня інтерфейсу наведена на рис. 3.2.

Рисунок 3.2 – Діаграма класів рівня інтерфейсу системи

Діаграма класів рівня інтерфейсу системи складається із наступних класів:

- LoginForm – форма для авторизації користувачів. Включає методи для перевірки введених даних, встановлення з’єднання з системою та обробки подій, таких як спроби входу;

- UsersForm – форма, яка надає інтерфейс для перегляду, додавання, редагування та видалення користувачів системи. Реалізує методи для роботи з таблицями користувачів, включаючи обробку клітинок і завантаження даних;

- ModelForm – інтерфейс для управління моделями машинного навчання. Дозволяє додавати, видаляти, переглядати та вибирати моделі для роботи. Реалізує функції обробки подій для взаємодії з моделями, включаючи їхнє завантаження;

- PredictForm – форма для роботи з прогнозами. Забезпечує взаємодію з користувачем для завантаження даних, генерації прогнозів за допомогою моделей машинного навчання та візуалізації результатів;

- PersonalizationForm – форма для налаштування індивідуальних параметрів користувача, таких як персоналізація доступу або вибір конфігурації;

- UpdateUsersForm – форма для редагування інформації про користувачів. Включає методи для збереження та оновлення даних у системі;

- SystemLogForm – інтерфейс для перегляду системних журналів. Дозволяє завантажувати та аналізувати дані про події системи.

- MonitoringMDI – головна форма моніторингу, що забезпечує доступ до різних функцій системи через меню. Включає методи для управління відкритими вікнами, доступу до журналів, роботи з прогнозами та іншими підсистемами.

Така структура рівня інтерфейсу дозволяє забезпечити зручний доступ до функцій системи, реалізувати розподіл завдань між окремими формами та інтегрувати інструменти машинного навчання для роботи з прогнозами й моніторингом.

Рівень бізнес-логіки системи реалізує основні функції для обробки даних, перевірки введеної інформації та підтримки ключових процесів у системі. Цей рівень забезпечує взаємодію між інтерфейсом користувача та рівнем даних, виконуючи проміжні обчислення, перевірки та інші операції. Діаграма класів рівня бізнес-логіки наведена на рис. 3.3.

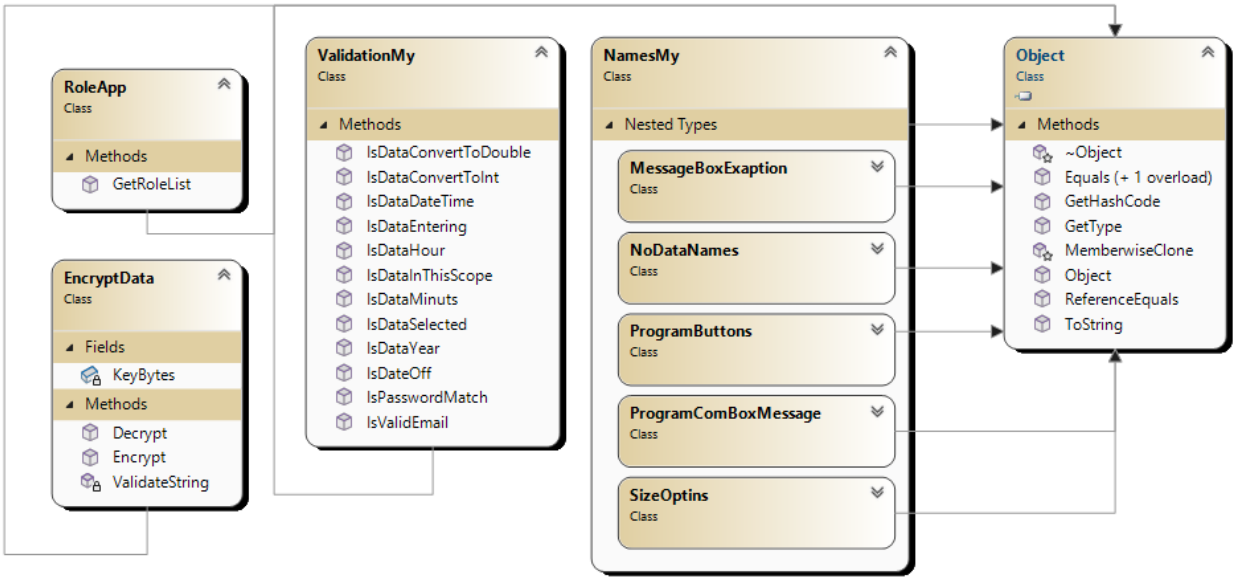


Рисунок 3.3 – Діаграма класів рівня бізнес-логіки

Діаграма класів цього рівня складається із:

- RoleApp – клас, що відповідає за управління ролями користувачів у системі. Основна функція цього класу – отримання списку ролей, що використовується для управління доступом до функцій системи;

- EncryptData – клас, призначений для шифрування та розшифрування даних. Містить поля для ключів шифрування, а також методи для перевірки рядків, шифрування та розшифрування інформації. Цей клас забезпечує безпеку передачі та зберігання даних у системі;

- ValidationMy – клас, що реалізує методи перевірки введених користувачем даних. Він включає функції для перевірки формату дат, чисел, електронних адрес, відповідності паролів тощо. Цей клас використовується для зменшення помилок введення та забезпечення коректності даних, що передаються на інші рівні системи;

- NamesMy – клас, що містить вкладені типи для представлення повідомлень, назв елементів інтерфейсу та стандартних значень. Серед вкладених типів є: MessageBoxExeption (обробка повідомлень про виключення), NoDataNames (обробка повідомлень про відсутність даних), ProgramButtons (зберігання

ідентифікаторів кнопок), ProgramComBoxMessage (обробка повідомлень комбобоксів), SizeOptins (представлення параметрів розміру).

Дана структура рівня бізнес-логіки забезпечує функціональну та структурну гнучкість системи, зменшуючи кількість помилок і підвищуючи безпеку при роботі з даними.

3.3 Завантаження та обробка датасету для тренування моделі

Побудова моделі машинного навчання починається з завантаження та обробки датасету, який служить основою для навчання алгоритму та оцінки його продуктивності. Цей процес включає кілька етапів, кожен з яких має своє значення для забезпечення якості та стабільності роботи моделі. Послідовна реалізація цих кроків дозволяє підготувати дані, обрати оптимальний алгоритм і застосувати модель у реальних умовах.

На рис. 3.4 наведена загальна структура побудови моделі машинного навчання, яка ілюструє ключові етапи роботи з даними, починаючи від їхньої підготовки та закінчуючи застосуванням отриманої моделі. Ця структура охоплює всі необхідні кроки для створення якісної моделі, здатної ефективно працювати в умовах реального часу.

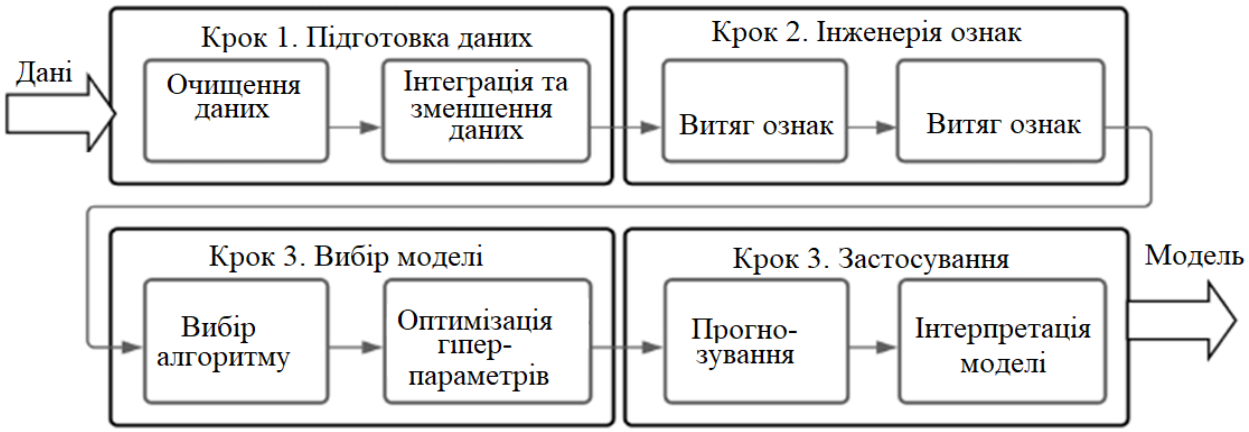


Рисунок 3.4 – Загальна структура побудови моделі машинного навчання

Процес побудови моделі починається з підготовки даних, яка включає очищення, інтеграцію та зменшення даних. Цей етап є критично важливим для усунення шуму, заповнення пропущених значень і видалення нерелевантної інформації, що впливає на подальшу точність моделі.

Другий етап передбачає інженерію ознак, яка зосереджена на виборі та конструюванні тих характеристик, які мають найбільше значення для моделювання. Цей крок дозволяє зменшити розмірність задачі, підвищити якість аналізу та покращити швидкість навчання моделей. Третій етап включає вибір алгоритму, оптимізацію його гіперпараметрів і навчання моделі. На цьому етапі відбувається підбір найбільш відповідного методу для аналізу даних, а також налаштування параметрів, які впливають на продуктивність моделі.

Завершальний етап стосується застосування моделі для прогнозування та інтерпретації результатів. У цьому кроці створена модель використовується для аналізу нових даних, а також для генерування інсайтів, які можна застосувати для подальшого вдосконалення процесів. Ця структура забезпечує комплексний підхід до роботи з даними та створення моделей машинного навчання, що робить її гнучкою та ефективною у вирішенні задач різного рівня складності.

На рис. 3.5 наведений DataFrame, сформований на основі промаркованих даних, взятих із авторитетного сайту kaggle [46].

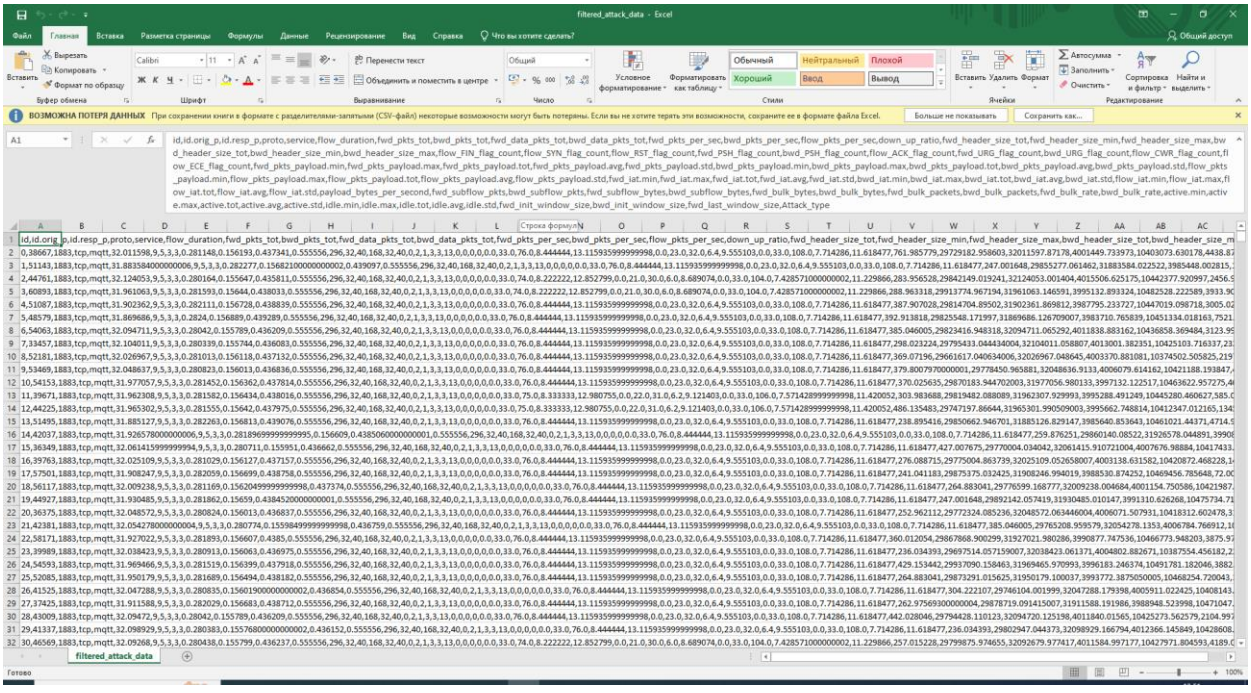


Рисунок 3.5 – Сформований датасет

Цей датасет містить численні записи з різними параметрами мережевого трафіку, які включають як числові, так і категоріальні ознаки, що характеризують поведінку системи. Ключова особливість цього набору даних – наявність міток, які

вказують на належність кожного запису до конкретного класу, наприклад, нормальної поведінки або аномалії (тип атаки).

У табл. 3.1 наведено опис ключових атрибутів датасету, що використовуються для навчання моделі. Кожен атрибут містить пояснення його ролі та значення в контексті аналізу мережевого трафіку. Це дозволяє не лише зрозуміти структуру даних, але й обґрунтувати їхню важливість у процесі моделювання.

Таблиця 3.1 – Опис атрибутів датасету

№	Атрибут	Опис
1	2	3
1	id	Ідентифікатор потоку або запису трафіку.
2	id_orig_p	Ідентифікатор оригінального джерела (адреса або інша інформація про джерело).
3	id_resp_p	Ідентифікатор цільового (респонсивного) кінцевого вузла в мережі.
4	proto	Протокол, який використовується в цьому потоці (наприклад, TCP, UDP).
5	service	Тип служби, що використовується в потоці (наприклад, HTTP, DNS).
6	flow_duration	Тривалість потоку в мілісекундах.
7	fwd_pkts_tot	Загальна кількість всіх пакетів, переданих від джерела (вперед) за потік.
8	bwd_pkts_tot	Загальна кількість всіх пакетів, отриманих від одержувача (назад) за потік.
9	fwd_data_pkts_tot	Кількість пакетів, що містять лише дані, передані від джерела (вперед).
10	bwd_data_pkts_tot	Кількість пакетів, що містять лише дані, отримані від одержувача (назад).
11	fwd_pkts_per_sec	Кількість пакетів, що відправляються вперед за секунду.
12	bwd_pkts_per_sec	Кількість пакетів, що надходять назад за секунду.
13	flow_pkts_per_sec	Кількість пакетів у потоці за секунду.
14	down_up_ratio	Відношення пакетів вниз до пакетів вгору.
15	fwd_header_size_tot	Загальний розмір заголовків, надісланих вперед.
16	fwd_header_size_min	Мінімальний розмір заголовка, надісланого вперед.
17	fwd_header_size_max	Максимальний розмір заголовка, надісланого вперед.

Продовження таблиці 3.1.

1	2	3
18	bwd_header_size_tot	Загальний розмір заголовків, надісланих назад.
19	bwd_header_size_min	Мінімальний розмір заголовка, надісланого назад.
20	bwd_header_size_max	Максимальний розмір заголовка, надісланого назад.
21	flow_FIN_flag_count	Кількість встановлених прапорців FIN у потоці (завершення TCP-з'єднання).
22	Attack_type	Тип атаки, який ідентифікується в цьому записі (мітка класу).

На рис. 3.6 представлена схема процесу побудови класифікаторів моделі машинного навчання та оцінки їхньої ефективності. Даний процес починається з формування набору даних, отриманого з мережевого трафіку, який після попередньої обробки розділяється на тренувальний і тестовий набори. Тренувальний набір використовується для навчання різних класифікаторів, а тестовий – для перевірки їхньої якості та точності.

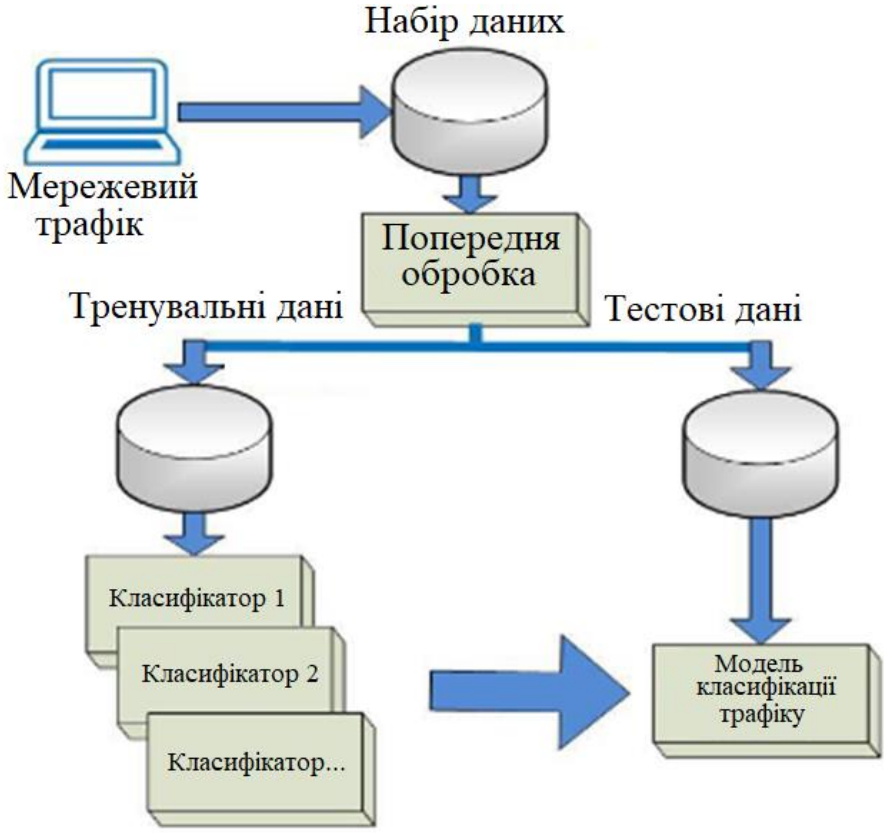


Рисунок 3.6 – Побудова класифікаторів моделі і валідації їх ефективності

Класифікатори, такі як дерева рішень, логістична регресія та метод опорних векторів, навчаються на тренувальних даних для аналізу мережевого трафіку та

класифікації його як нормального або такого, що містить атаки. В процесі навчання виконуються оптимізація гіперпараметрів і тестування моделей на основі ряду ефективних показників, таких як точність (Precision), повнота (Recall) і F1-оцінка. Оцінювання на тестовому наборі дозволяє визначити продуктивність моделей і їхню здатність до генералізації.

Готові моделі зберігаються у відповідному форматі для подальшого використання у реальному часі. Це значно скорочує ресурси, необхідні для повторного навчання, і дозволяє ефективно аналізувати нові дані, зокрема в умовах динамічних змін у мережевому середовищі.

3.4 Реалізація коду системи моніторингу інфраструктури

Реалізація системи моніторингу інфраструктури базується на інтеграції функціоналу для роботи з даними, їхньої обробки та представлення користувачу. Одним із ключових аспектів реалізації є забезпечення можливості завантаження файлів із вхідними даними, які використовуються для аналізу та роботи моделей машинного навчання (рис. 3.7). Такий підхід дозволяє гнучко адаптувати систему під різні сценарії моніторингу та аналітики.

```
private async void OpenBtn_Click(object sender, EventArgs e) {
  // Створення діалогового вікна для відкриття файлу
  OpenFileDialog openFileDialog = new OpenFileDialog();

  // Налаштування властивостей діалогового вікна
  openFileDialog.Filter = "Text files (*.csv)|*.csv|All files (*.*)|*.*";
  openFileDialog.FilterIndex = 2;
  openFileDialog.RestoreDirectory = true;

  // Відображення діалогового вікна та обробка результату
  if (openFileDialog.ShowDialog() == DialogResult.OK) {
    _Path = openFileDialog.FileName;
    FileNameTBox.Text = openFileDialog.FileName;
```

Рисунок 3.7 – Завантаження файлів із вхідними даними

Важливим компонентом є механізм вибору файлів із системи. Для цього в системі реалізовано інтерактивний інтерфейс, що дозволяє користувачеві завантажувати файли в форматі CSV або інших підтримуваних форматах. Після активації кнопки вибору файлу відкривається діалогове вікно, яке дає змогу обрати потрібний файл. Параметри діалогового вікна включають фільтрацію за типом

файлів, збереження останнього вибраного шляху та забезпечення зручності використання.

Після вибору файлу його шлях автоматично передається системі для подальшої обробки. Цей підхід дає змогу підключати дані для аналізу без необхідності додаткових налаштувань з боку користувача, що спрощує процес моніторингу та підвищує його ефективність. Така функціональність є невід'ємною частиною системи, що дозволяє інтегрувати роботу з різними джерелами даних та забезпечує гнучкість використання.

У процесі реалізації системи моніторингу одним із важливих кроків є створення контексту машинного навчання, який забезпечує інтеграцію з інструментами для обробки даних і побудови моделей (рис. 3.8). У коді створюється об'єкт MLContext із заданим початковим значенням для генератора випадкових чисел, що гарантує відтворюваність результатів під час навчання моделей.

```
mlContext = new MLContext(seed: 0);

// Завантаження даних
dataView = mlContext.Data.LoadFromTextFile<InputData>(_Path,
    hasHeader: true,
    separatorChar: ',');
```

Рисунок 3.8 – Створення контексту навчання та завантаження даних

За допомогою методу LoadFromTextFile завантажується файл із вхідними даними до структури класу InputData. Параметри завантаження вказують на наявність заголовків у файлі та визначають символ, який використовується для розділення значень у записах. Цей процес дозволяє підготувати дані для подальшої обробки та аналізу, автоматизуючи їх інтеграцію в конвеєр навчання моделі машинного навчання.

У фрагменті коду на рис. 3.9 реалізується обробка завантажених даних для аналізу розподілу типів атак у датасеті. За допомогою методу CreateEnumerable дані з файлу перетворюються у формат, придатний для роботи в коді як колекція об'єктів класу InputData. Подальше групування здійснюється на основі атрибута Attack_type, що дозволяє визначити, скільки записів відповідає кожному типу атаки.

```
var attackTypeCounts =
  mlContext.Data.CreateEnumerable<InputData>(dataView, reuseRowObject: false)
    .GroupBy(x => x.Attack_type)
    .Select(g => new { AttackType = g.Key, Count = g.Count() })
    .OrderByDescending(g => g.Count);

foreach (var item in attackTypeCounts) {
  RaportTBox.Text += ($"{item.AttackType} - {item.Count}\r\n");
}
```

Рисунок 3.9 – Створення контексту навчання та завантаження даних

Для кожної групи розраховується кількість записів, після чого ці дані сортуються за спаданням, щоб типи атак з найбільшою частотою знаходилися на початку списку. Результати обробки послідовно додаються до текстового поля RaportTBox, що надає користувачеві зручний формат для перегляду статистики про кількість записів для кожного типу атаки.

На рис. 3.10 наведено код підготовки даних для навчання моделі – розділення на тренувальний і тестовий набори. Використовується метод TrainTestSplit, який дозволяє поділити початковий набір даних на дві частини: більшу для навчання моделі та меншу для оцінки її продуктивності. У цьому випадку 20% даних виділяється для тестування, що є стандартним підходом у задачах машинного навчання.

```
var trainTestSplit = mlContext.Data.TrainTestSplit(dataView, testFraction: 0.2);
```

Рисунок 3.10 – Розділення даних на тренувальний та тестовий набори

У коді на рис. 3.11 реалізується підготовка даних для машинного навчання шляхом створення конвеєра обробки, який включає декілька етапів трансформацій. Спочатку проводиться кодування категоріальних ознак, таких як протокол (proto) і тип служби (service), у формат OneHotEncoding, який перетворює їх у числові вектори, зручні для роботи з моделями. Це дозволяє моделі коректно інтерпретувати текстові дані. Далі за допомогою методу Concatenate всі числові ознаки, включаючи закодовані категоріальні дані, об’єднуються у вектор характеристик під назвою Features. Цей етап забезпечує формування єдиного набору входів для моделі, що містить всі важливі параметри мережевого трафіку.

```
var dataProcessPipeline = mlContext.Transforms.Categorical.OneHotEncoding(
    outputColumnName: "protoEncoded", inputColumnName: "proto")
.Append(mlContext.Transforms.Categorical.OneHotEncoding(
    outputColumnName: "serviceEncoded", inputColumnName: "service"))
.Append(mlContext.Transforms.Concatenate("Features",
    // Числові ознаки
    nameof(InputData.id), nameof(InputData.id_orig_p),
    nameof(InputData.id_resp_p), nameof(InputData.flow_duration),
    nameof(InputData.fwd_pkts_tot), nameof(InputData.bwd_pkts_tot),
    nameof(InputData.fwd_data_pkts_tot), nameof(InputData.bwd_data_pkts_tot),
    nameof(InputData.fwd_pkts_per_sec),  nameof(InputData.bwd_pkts_per_sec),
    nameof(InputData.flow_pkts_per_sec), nameof(InputData.down_up_ratio),
    nameof(InputData.fwd_header_size_tot), nameof(InputData.fwd_header_size_min),
    nameof(InputData.fwd_header_size_max), nameof(InputData.bwd_header_size_tot),
    nameof(InputData.bwd_header_size_min), nameof(InputData.bwd_header_size_max),
    nameof(InputData.flow_FIN_flag_count),
    "protoEncoded",
    "serviceEncoded"))
.Append(mlContext.Transforms.ReplaceMissingValues(
    inputColumnName: "Features", outputColumnName: "Features"))
.Append(mlContext.Transforms.NormalizeMinMax("Features"))
.Append(mlContext.Transforms.Conversion.MapValueToKey("Label", "Attack_type"));
```

Рисунок 3.11 – Створення конвеєра обробки даних

Щоб підвищити якість даних, додається заміна відсутніх значень, яка гарантує, що пропущені дані не впливатимуть негативно на результати. Після цього проводиться нормалізація значень у векторі Features за допомогою методу NormalizeMinMax, що вирівнює діапазони значень і забезпечує стабільність навчання. На завершення мітки класу Attack_type перетворюються у числові ключі, що адаптує вихідні дані для задачі класифікації. Такий багатоступеневий підхід гарантує якісну обробку даних перед подачею їх у модель.

Фрагмент коду на рис. 3.12 реалізує етап вибору алгоритму навчання та безпосереднього тренування моделі.

```
var trainer =
  mlContext.MulticlassClassification.Trainers.LbfgsMaximumEntropy("Label", "Features")
    .Append(mlContext.Transforms.Conversion.MapKeyToValue("PredictedLabel"));
var trainingPipeline = dataProcessPipeline.Append(trainer);
// Навчання моделі
model = trainingPipeline.Fit(trainTestSplit.TrainSet);
```

Рисунок 3.12 – Вибір алгоритму та тренування моделі

Для задачі багатокласової класифікації використовується алгоритм LbfgsMaximumEntropy, який забезпечує ефективне моделювання залежностей між ознаками та мітками класів. Алгоритм використовує метод максимізації ентропії, що дозволяє працювати з багатокласовими даними з високою точністю. До обраного алгоритму додається крок, який перетворює передбачувані числові

значення назад у текстові мітки класів за допомогою MapKeyToValue. Ця трансформація дозволяє моделі видавати результати у зручному для інтерпретації вигляді.

Після визначення тренера модель будується шляхом об'єднання обробки даних, налаштованої раніше у конвеєрі, із самим алгоритмом навчання. Завершальний етап – навчання моделі на тренувальному наборі даних, який автоматично оптимізує параметри моделі для досягнення максимальної точності класифікації. Отримана модель готова для застосування на нових даних.

У коді на рис. 3.13 здійснюється оцінка продуктивності моделі машинного навчання на основі тестового набору даних. Першим кроком є застосування навченої моделі до тестового набору для отримання прогнозів. Ці перетворені дані містять як реальні мітки класів, так і передбачені значення, що дозволяє здійснити подальший аналіз ефективності моделі.

```
var predictions = model.Transform(trainTestSplit.TestSet);
// Оцінюємо модель
var metrics = mlContext.MulticlassClassification.Evaluate(predictions);
// Заголовок для метрик
RaportTBox.Text += ("Метрики для кожного типу вразливостей:\r\n");
// Отримуємо матрицю плутанини
var confusionMatrix = metrics.ConfusionMatrix;
// Отримуємо кількість класів
int classCount = confusionMatrix.NumberOfClasses;
```

Рисунок 3.13 – Оцінка продуктивності моделі машинного навчання

Також виконується обчислення метрик за допомогою вбудованого методу Evaluate, який аналізує точність, повноту, F1-оцінку та інші ключові показники для багатокласової класифікації. Результати включають загальні характеристики продуктивності моделі, що є основою для оцінки її якості. Для детального аналізу використовується матриця плутанини, яка візуалізує співвідношення між реальними мітками класів і прогнозованими значеннями. Цей підхід дозволяє оцінити коректність класифікації для кожного класу окремо. Додатково визначається кількість класів у задачі, що необхідно для подальшої роботи з метриками або оптимізації моделі. Отримані результати відображаються у текстовому полі, надаючи користувачеві інформативну оцінку ефективності моделі.

Код на рис. 3.14 спрямований на аналіз моделі, зокрема на оцінку її продуктивності для кожного класу. Спочатку з метаданих стовпця Score витягуються назви класів, що дозволяє ідентифікувати кожен клас, для якого проводиться класифікація. Це досягається через використання методу GetSlotNames, який повертає імена класів у вигляді колекції, а потім вони перетворюються у зручний для роботи масив рядків.

```
// Отримуємо назви класів з метаданих стовпця 'Score'
VBuffer<ReadOnlyMemory<char>> slotNames = default;
predictions.Schema["Score"].GetSlotNames(ref slotNames);
var classNames = slotNames.DenseValues().Select(x => x.ToString()).ToArray();
// Отримуємо матрицю помилок як 2D-масив
var counts = confusionMatrix.Counts;
// Масиви для зберігання метрик
double[] perClassPrecision = new double[classCount];
double[] perClassRecall = new double[classCount];
double[] perClassF1Score = new double[classCount];
```

Рисунок 3.14 – Оцінка продуктивності для кожного класу моделі

Далі з матриці плутанини, отриманої на попередньому етапі, витягується двовимірний масив значень, який відображає кількість правильних і неправильних класифікацій для кожного класу. Ця інформація є основою для розрахунку метрик точності, повноти та F1-оцінки для кожного з класів. Окремі масиви створюються для зберігання метрик, таких як precision, recall та F1-оцінка для кожного класу. Цей підхід дозволяє структурувати дані та підготувати їх до подальшого аналізу чи візуалізації, що є важливим для розуміння сильних і слабких сторін моделі. Розраховані метрики надають детальну картину продуктивності класифікатора та допомагають визначити, які класи потребують додаткового навчання або корекції моделі.

Код на рис. 3.15 обчислює ключові метрики ефективності класифікації для кожного класу, використовуючи дані з матриці плутанини. У циклі, що проходить через усі класи, визначаються істинно позитивні (True Positive) значення, які відповідають коректно класифікованим прикладам для кожного класу. Додатково розраховуються помилково позитивні (False Positive) та помилково негативні (False Negative) значення для кожного класу, що дозволяє повністю охопити всі можливі ситуації неправильної класифікації.

```
// Цикл для обчислення метрик по кожному класу
for (int i = 0; i < classCount; i++) {
  double truePositive = counts[i][i];
  double falsePositive = 0;
  double falseNegative = 0;
  // Обчислюємо False Positive та False Negative для кожного класу
  for (int j = 0; j < classCount; j++) {
    if (i != j) {
      falsePositive += counts[j][i];
      falseNegative += counts[i][j];
    }
  }
  // Обчислюємо Precision, Recall та F1 Score
  perClassPrecision[i] = truePositive / (truePositive + falsePositive + 1e-6);
  perClassRecall[i] = truePositive / (truePositive + falseNegative + 1e-6);
  perClassF1Score[i] = 2 * perClassPrecision[i] * perClassRecall[i] /
                       (perClassPrecision[i] + perClassRecall[i] + 1e-6);
  // Виводимо метрики для кожного класу
  string className = classNames.Length > i ? classNames[i] : $"Клас {i}";
  RaportTBox.Text += ($"Тип вразливості: {className}\r\n");
  RaportTBox.Text += ($"  Precision: {perClassPrecision[i]:P2}\r\n");
  RaportTBox.Text += ($"  Recall: {perClassRecall[i]:P2}\r\n");
  RaportTBox.Text += ($"  F1 Score: {perClassF1Score[i]:P2}\r\n");
}
```

Рисунок 3.15 – Оцінка продуктивності моделі машинного навчання

На основі цих значень обчислюються три важливі метрики: точність (Precision), повнота (Recall) та F1-оцінка. Для запобігання помилкам, пов'язаним із поділом на нуль, додається невелике значення (1e-6) у знаменник формул. Precision оцінює, наскільки коректними є передбачення для даного класу, Recall відображає здатність моделі виявляти всі приклади цього класу, а F1-оцінка є гармонійним середнім цих двох метрик, забезпечуючи збалансовану оцінку продуктивності.

Результати для кожного класу, включаючи назву класу та значення метрик, додаються до текстового поля RaportTBox. Такий підхід дозволяє отримати зрозумілий і деталізований звіт про продуктивність моделі для кожного класу, що є важливим для аналізу її роботи та подальшого вдосконалення.

Рис. 3.16 відображає код, що реалізує функціонал збереження навченої моделі, а також фіксацію цього процесу в системних журналах. Після перевірки правильності введених даних методом IsDataEnteringCorrect виконується створення унікального імені файлу для моделі. Шлях для збереження файлу генерується з урахуванням поточного місця розташування програми, забезпечуючи структуроване зберігання моделей у визначеній директорії.

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

Рисунок 3.16 – Метод збереження моделі

Інформація про нову модель додається до бази даних через метод _ModelsProvider.InsertModels, що дозволяє зберігати назву моделі та шлях до її файлу. Далі модель зберігається на диску за допомогою вбудованого методу mlContext.Model.Save, який враховує структуру даних для її подальшого використання. Після завершення збереження даних поля введення очищуються, що забезпечує зручність роботи для користувача. Для аудиту та відстеження дій користувача в систему журналів записується інформація про створення нової моделі, зокрема її назва та час операції, використовуючи _LogsProvider.InsertLogs. На завершення користувач отримує повідомлення про успішне збереження даних, що забезпечує інтуїтивну взаємодію з програмою.

3.5 Тренування моделі та її оцінка

Процес тренування моделі машинного навчання є ключовим етапом у створенні інтелектуальної системи моніторингу. Він базується на добре підготовленому датасеті, який забезпечує достатню якість і репрезентативність даних для побудови ефективної класифікації. Формування навчальної вибірки передбачає очищення, фільтрацію, балансування даних і забезпечення їх відповідності задачі багатокласової класифікації. Такий підхід дозволяє моделі коректно навчатися на реальних прикладах, що імітують сценарії роботи мережевої інфраструктури.

Для тренування використовувався датасет, що включає записи про різні типи атак (табл. 3.2). Після попередньої обробки з нього було видалено дублікати,

некоректні та неповні записи, а також виконано балансування вибірки, щоб уникнути переваги одних типів атак над іншими. Це гарантує, що модель отримає рівноцінну кількість даних для кожного класу, зменшуючи ризик її зміщення.

Таблиця 3.2 – Кількість записів по кожному типу атаки

Тип атаки	Кількість записів
DOS_SYN_Hping	10,000
Thing_Speak	8,108
ARP_poisioning	7,750
MQTT_Publish	4,146
NMAP_UDP_SCAN	2,590
NMAP_XMAS_TREE_SCAN	2,010
NMAP_OS_DETECTION	2,000
NMAP_TCP_scan	1,002
DDOS_Slowloris	534
Wipro_bulb	253
Metasploit_Brute_Force_SSH	37

Датасет охоплює різноманітні атаки, включаючи DoS, ARP-підробки, сканування портів і SSH-брутфорс, що дозволяє моделі навчатися на реальних сценаріях. Балансування даних та рівномірний розподіл записів створюють основу для побудови стійкої моделі, здатної до коректного розпізнавання атак у реальних умовах.

На основі підготовленого датасету було виконано навчання моделі машинного навчання для класифікації різних типів атак у мережевій інфраструктурі. Датасет, що включає записи про широке коло атак, дозволив забезпечити якісне тренування, враховуючи всі суттєві аспекти кожного типу загроз. Використані алгоритми забезпечили ефективну обробку особливостей вхідних даних, що позитивно вплинуло на результати навчання.

Рис. 3.17 ілюструє процес навчання моделі, підкреслюючи важливість збалансованого набору даних та врахування різноманітних сценаріїв атак для досягнення стабільних і точних результатів.

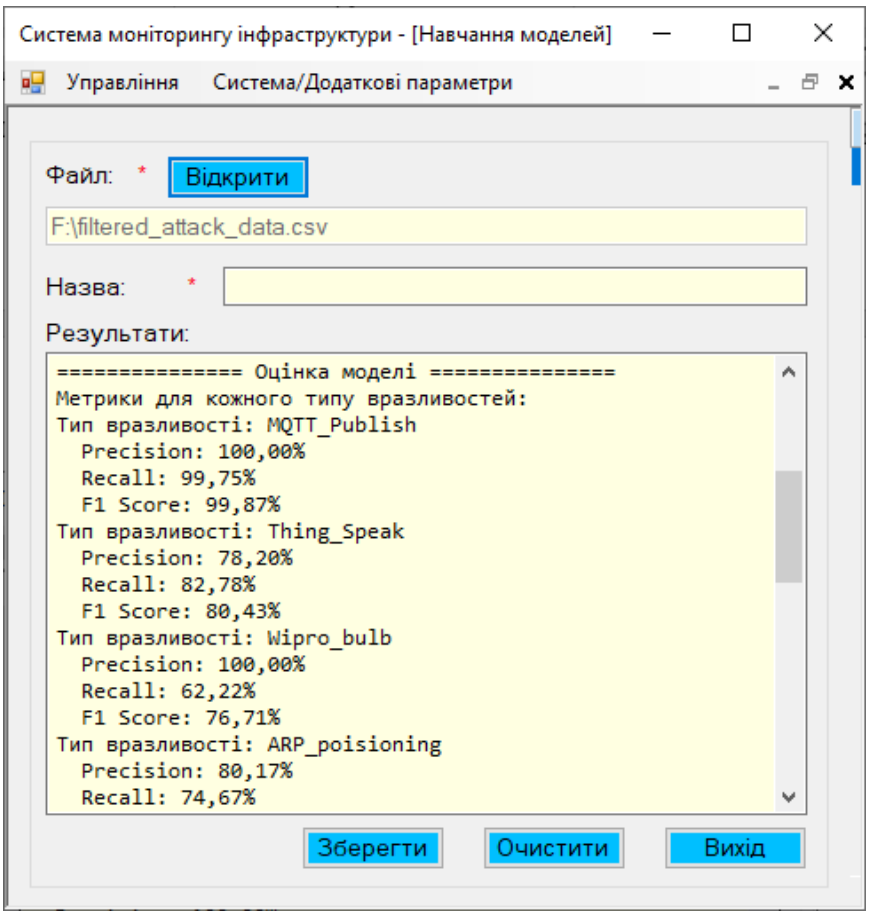


Рисунок 3.17 – Процес навчання моделі

Результати тренування моделі та оцінки її точності наведені на рис. 3.18, який ілюструє точність класифікації для кожного типу вразливостей. Графік демонструє розподіл точності у відсотках для різних атак, включаючи DoS, ARP-poisoning, сканування NMAP, MQTT та інші.

З графіка видно, що для більшості атак модель досягає високої точності, наближеної до 100%. Зокрема, класифікація атак NMAP_TCP_scan, DOS_SYN_Hping та NMAP_OS_DETECTION показала чудові результати, що вказує на здатність моделі коректно розпізнавати ці види загроз. Водночас деякі типи атак, такі як Wipro_bulb та Metasploit_Brute_Force_SSH, мають нижчі показники, що може бути наслідком меншої кількості записів у датасеті для цих класів.

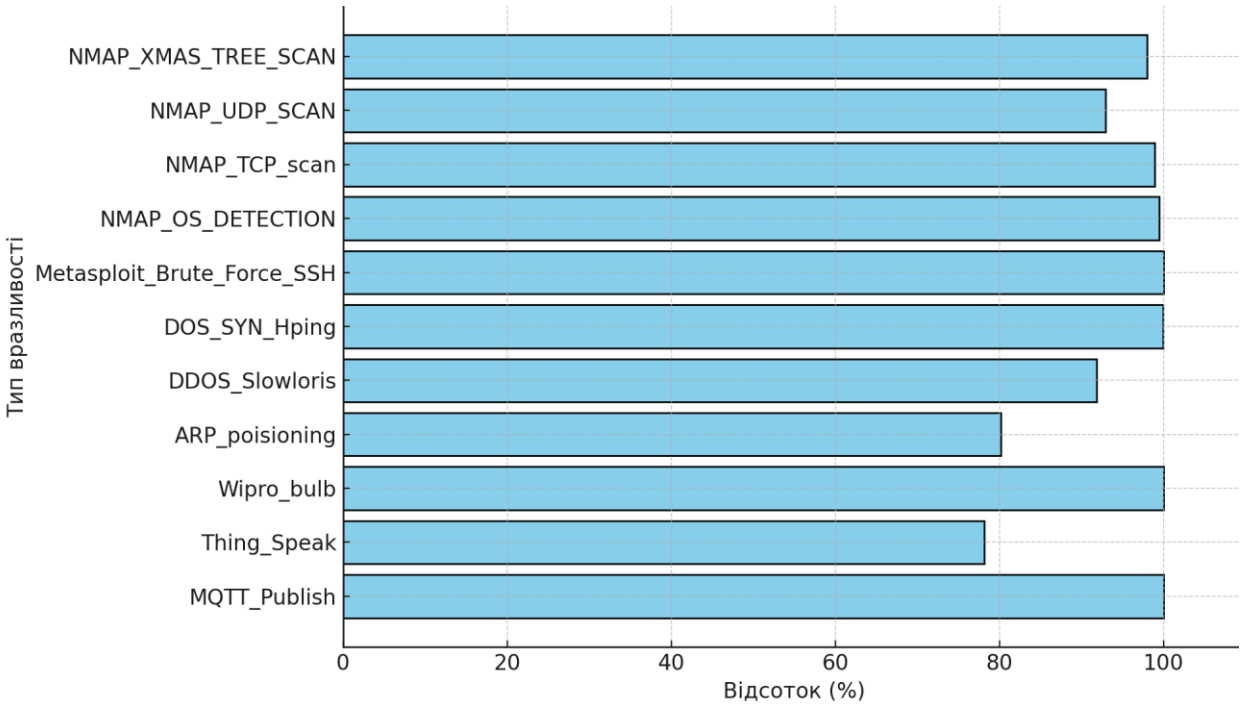


Рисунок 3.18 – Точність для кожного типу вразливості

Загалом аналіз точності підтверджує ефективність моделі для більшості атак, однак результати для менш поширених типів вразливостей вказують на необхідність додаткового збору даних або уточнення параметрів моделі. Це дозволить покращити її продуктивність та забезпечити більш стабільне розпізнавання рідкісних загроз.

Результати оцінки повноти класифікації для кожного типу вразливостей представлені на рис. 3.19. Графік демонструє, наскільки ефективно модель виявляє всі зразки, що належать до певного класу. Повнота є важливим показником, оскільки вона характеризує здатність моделі правильно визначати всі позитивні приклади для кожного типу атаки.

Аналіз графіка показує, що повнота для таких типів атак, як NMAP_XMAS_TREE_SCAN, NMAP_TCP_scan, DOS_SYN_Hping та NMAP_OS_DETECTION, досягає майже 100%, що свідчить про високу здатність моделі розпізнавати ці загрози без втрати важливих даних. Водночас атаки, такі як Wipro_bulb і Metasploit_Brute_Force_SSH, демонструють нижчі значення повноти, що, ймовірно, пов’язано з обмеженою кількістю записів у датасеті для цих класів.

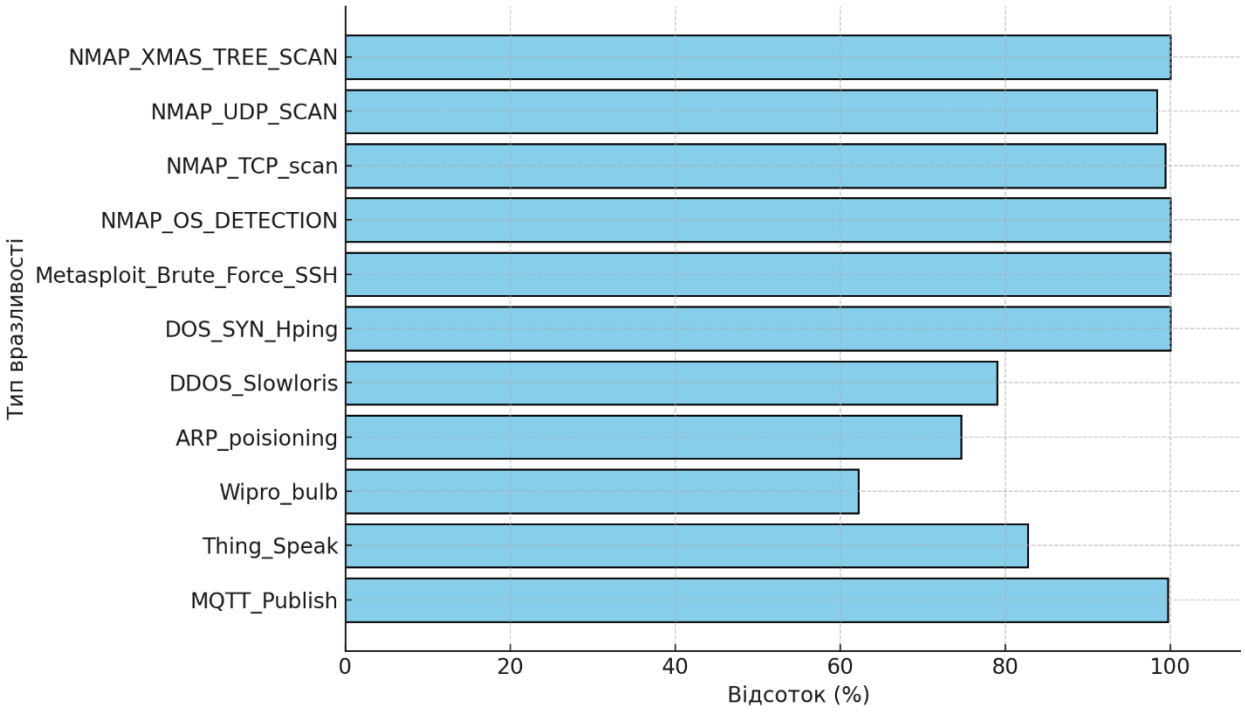


Рисунок 3.19 – Повнота для кожного типу вразливості

Загалом, результати підтверджують, що модель добре справляється із завданням розпізнавання поширених типів атак, однак для менш представлених класів необхідно проводити додаткове доопрацювання, зокрема збільшення вибірки або адаптацію параметрів моделі. Це сприятиме забезпеченню рівномірної повноти для всіх класів.

Результати оцінки F1-метрики для кожного типу вразливостей відображені на рис. 3.20. F1-оцінка є гармонійним середнім між точністю та повнотою, що дозволяє отримати збалансовану характеристику продуктивності моделі. Ця метрика є особливо важливою в умовах нерівномірного розподілу класів, оскільки враховує як правильність класифікації, так і здатність виявляти всі зразки.

Аналіз графіка демонструє, що модель досягає високих значень F1-оцінки для атак, таких як NMAP_XMAS_TREE_SCAN, DOS_SYN_Hping та NMAP_TCP_scan, що свідчить про її високу ефективність у класифікації цих загроз. Водночас для менш представлених атак, таких як Wipro_bulb та Metasploit_Brute_Force_SSH, значення F1-метрики є нижчими, що вказує на труднощі з їх розпізнаванням, зумовлені обмеженим обсягом даних у датасеті.

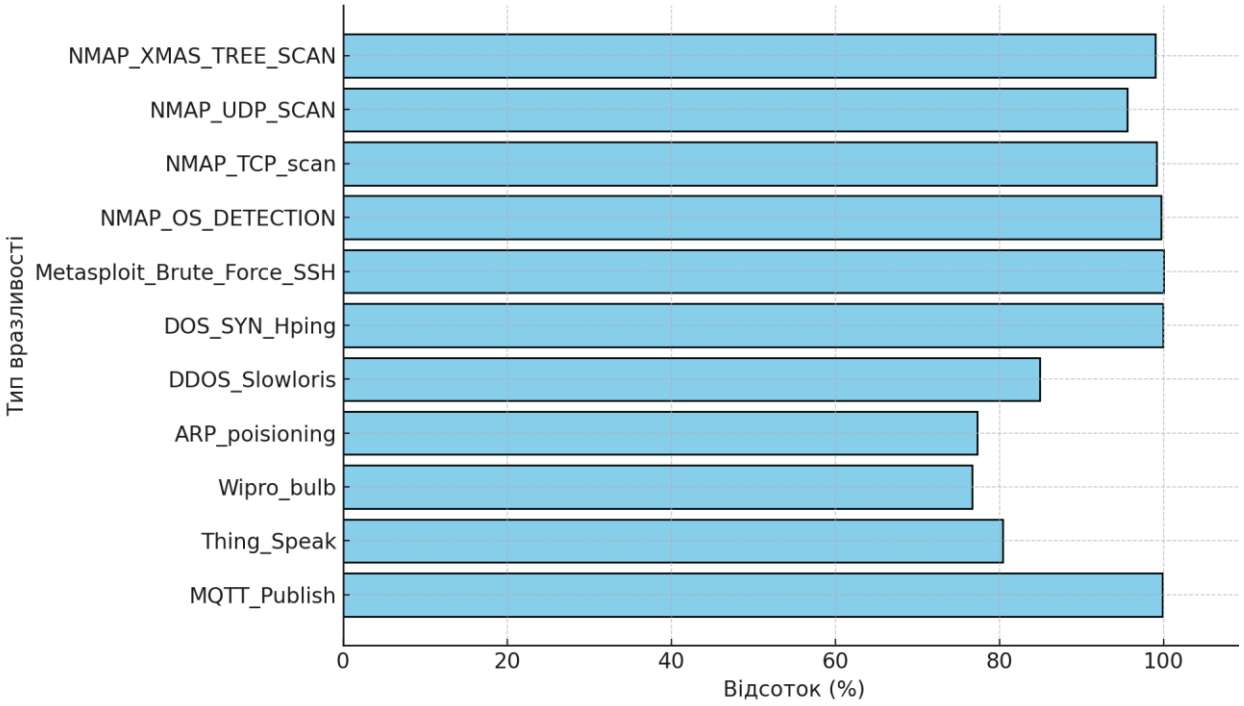


Рисунок 3.20 – F1-оцінка для кожного типу вразливості

Таким чином, F1-метрика підтверджує сильні сторони моделі у виявленні поширених атак, але також підкреслює необхідність вдосконалення для менш поширених класів, наприклад, шляхом збільшення обсягу даних чи впровадження додаткових механізмів для балансування вибірки. Це дозволить підвищити загальну стабільність та універсальність моделі.

3.6 Приклади та результати виявлення та обробки аномалій у реальних умовах

Після етапу навчання моделі та оцінки її основних характеристик, проведено тестування системи в реальних сценаріях для перевірки здатності до виявлення мережевих аномалій. Експерименти були спрямовані на оцінку точності прогнозування, адаптивності до різних типів даних та швидкості обробки вхідної інформації. В рамках цих сценаріїв система працювала з потоком даних, що імітують реальні умови експлуатації мережевої інфраструктури, для перевірки її ефективності у класифікації аномалій.

В першому сценарії тестування система отримала мережевий потік із наступними характеристиками: протокол TCP, служба DNS, тривалість потоку 150

мс, кількість пакетів від джерела та одержувача по одному, а швидкість передачі даних для обох напрямків становила 1677 пакетів за секунду (рис. 3.21). Загальний розмір заголовків у потоці був ідентичним для обох напрямків – 20 байтів. Відсутність прапорців FIN у потоці свідчить про нехарактерний завершення з'єднання, що могло слугувати однією з ознак атаки.

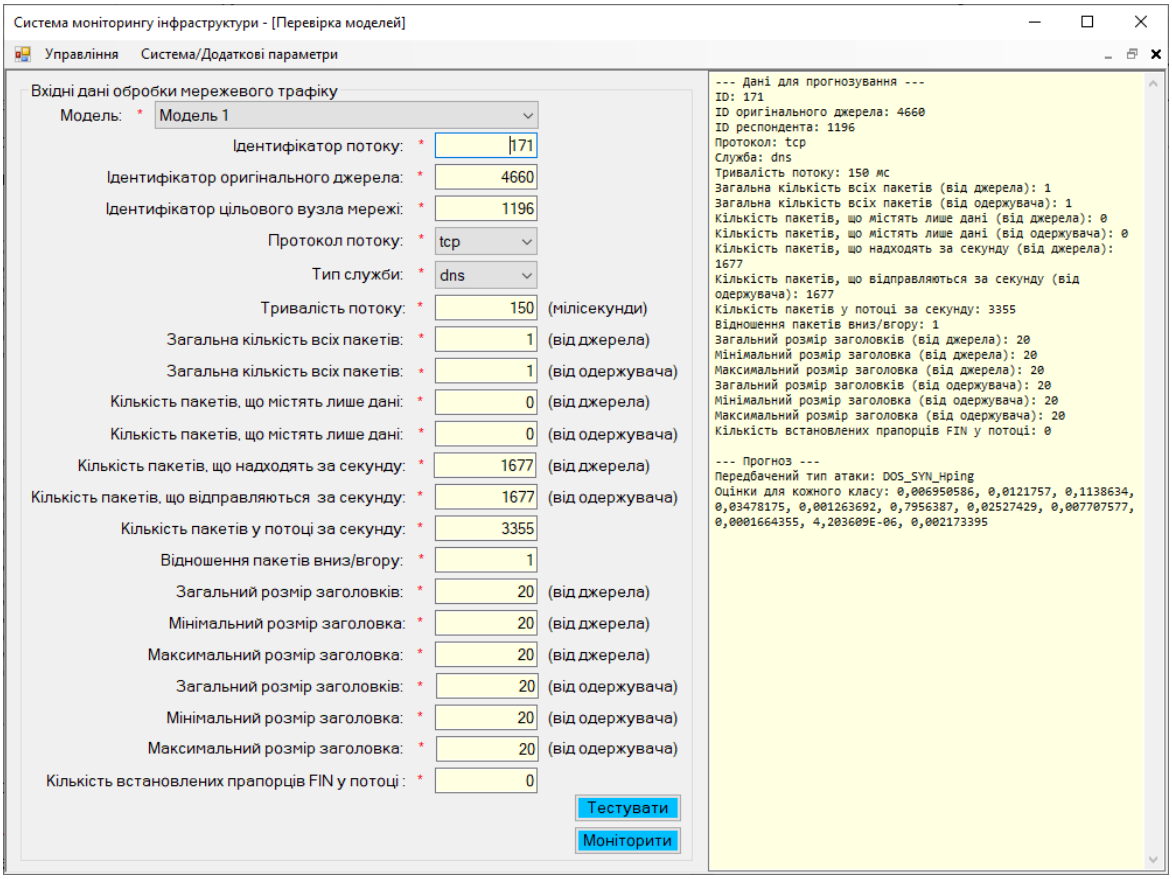


Рисунок 3.21 – Тестування системи для 1-го сценарію

На основі аналізу цих даних модель передбачила тип атаки DOS_SYN_Hping, з ймовірністю 79.56%. Висока ймовірність вказує на те, що модель впевнено ідентифікувала цей потік як частину атаки типу DoS, характерної для SYN-флуду. Такі атаки спрямовані на виснаження ресурсів системи шляхом надсилання великої кількості підроблених запитів. Додатково були розраховані оцінки для інших класів, серед яких найближча альтернатива (NMAP_TCP_scan) мала лише 11.38%, що підтверджує домінуючу впевненість моделі в обраному прогнозі.

Результати показують, що модель не лише коректно ідентифікує тип атаки, але й демонструє високу чутливість до специфічних характеристик потоку. Це підкреслює її придатність для роботи в реальних умовах, забезпечуючи як

швидкість, так і точність аналізу. У подальшому такі результати можуть бути використані для автоматизованого реагування на загрози, таких як блокування підозрілих потоків або зміна конфігурації мережевих пристроїв.

На рис. 3.22 наведено результати тестування 2-го сценарію.

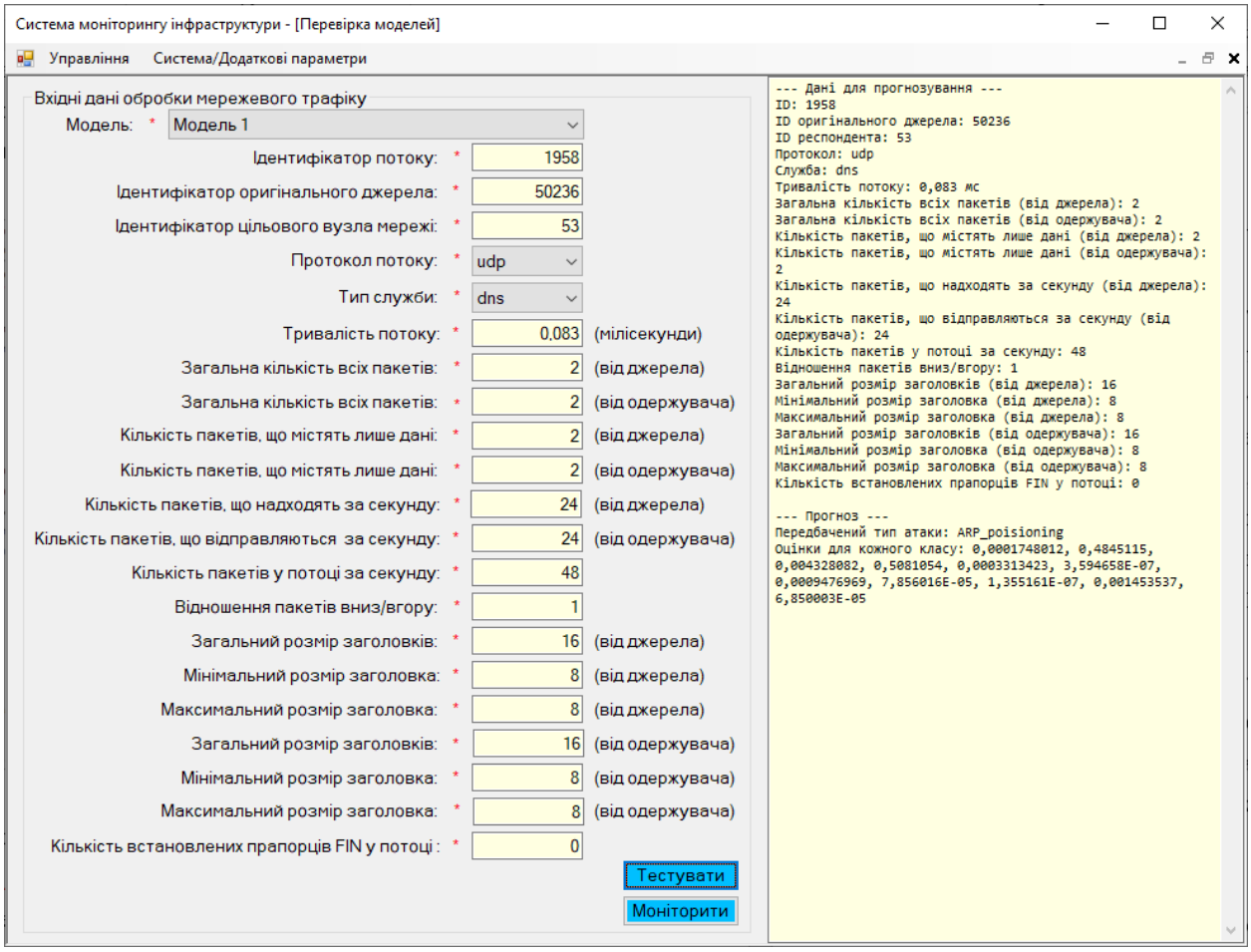


Рисунок 3.22 – Тестування системи для 2-го сценарію

У другому сценарії тестування система отримала потік із характеристиками, які дозволяють оцінити її здатність до розпізнавання специфічних аномалій у мережевому трафіку. Потік характеризувався використанням протоколу UDP, службою DNS, тривалістю 83 мс, двома пакетами від джерела та одержувача, а також кількістю пакетів за секунду у потоці, що становила 48. Розміри заголовків для обох напрямків дорівнювали 16 байтам, а кількість прапорців FIN була відсутня.

Результатом аналізу цього потоку стало передбачення типу атаки ARP_poisioning із найвищою оцінкою ймовірності серед інших класів. Модель

визначила цей тип атаки з упевненістю 48.45%, що значно перевищує оцінки для інших класів, таких як DOS_SYN_Hping чи MQTT_Publish.

Висока ймовірність прогнозу підтверджує здатність системи розпізнавати ARP-атаки на основі специфічних характеристик трафіку, таких як кількість пакетів, розміри заголовків і співвідношення напрямків передачі даних. Тестування також продемонструвало швидкість аналізу потоку, яка дозволяє використовувати модель у реальному часі.

Результати цього сценарію також підкреслюють ефективність системи у виявленні атак, що впливають на протоколи нижніх рівнів, таких як ARP. Однак можливе підвищення точності може бути досягнуто шляхом подальшого уточнення моделі чи додаткової оптимізації параметрів для менш виразних потоків.

ВИСНОВКИ

Дана робота присвячена дослідженню застосування сучасних методів машинного навчання для виявлення та обробки аномалій у системах моніторингу інфраструктури. В рамках роботи були розглянуті теоретичні основи, обрано технологічний стек та реалізовано систему, яка забезпечує аналіз мережевого трафіку та виявлення атак із використанням багатокласової класифікації. Досягнуті результати підтверджують ефективність розробленого підходу та дозволяють зробити висновки про доцільність інтеграції штучного інтелекту в системи моніторингу.

У першому розділі було детально проаналізовано існуючі рішення для моніторингу інфраструктури, зокрема системи Nagios XI, ManageEngine OpManager, Paessler PRTG та Zabbix. Було виділено їх переваги та недоліки, що дозволило сформувати базу для вибору оптимального підходу до розробки. Крім того, висвітлено методи обробки та аналізу даних, роль класифікації у моніторингу, а також принципи застосування машинного навчання для виявлення атак, що заклало фундамент для подальшої реалізації проєкту.

Другий розділ роботи присвячено вибору технологічного стеку та методології розробки. Було обґрунтовано вибір мови програмування C# і платформи ML.NET для реалізації системи, а також MS SQL Server для управління даними. Методологія розробки включала створення алгоритмів навчання моделі та моніторингу трафіку, що дозволило забезпечити інтеграцію всіх компонентів системи в єдину архітектуру.

У третьому розділі було розглянуто практичні аспекти реалізації системи. Постановка задачі виявлення атак базувалася на використанні багатокласової класифікації, а проектування трирівневої архітектури забезпечило гнучкість і масштабованість системи. Було завантажено та оброблено великий набір даних, що включав різні типи атак. Створена модель пройшла етапи навчання та оцінки, демонструючи високі метрики точності, повноти та F1-оцінки для більшості типів атак. Наприклад, для Metasploit_Brute_Force_SSH та DOS_SYN_Hping модель

досягла 100% точності й F1-оцінки. Результати тестових сценаріїв підтвердили здатність системи швидко й ефективно ідентифікувати загрози в реальних умовах, забезпечуючи аналіз потоків за мілісекунди.

Проведена робота демонструє, що застосування методів машинного навчання дозволяє суттєво покращити виявлення атак у системах моніторингу інфраструктури. Розроблений додаток забезпечує точність класифікації, високу швидкість обробки та можливість адаптації до різних сценаріїв. Отримані результати можуть бути використані для автоматизації процесів моніторингу, підвищення безпеки мережевих інфраструктур і створення передумов для розширення функціональності системи у майбутньому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. I. H. Sarker, H. Janicke, M. A. Ferrag, and A. Abuadbba. Multi-aspect rule-based ai: Methods, taxonomy, challenges and directions toward automation, intelligence and transparent cybersecurity modeling for critical infrastructures. Internet of Things, 23 pages. p. 101110, 2024. (дата звернення: 14. 10. 2024).

2. Hughes Jack, Pastrana Sergio, Hutchings Alice, Afroz Sadia, Samtani Sagar, Li Weifeng, and Ericsson Santana Marin. (2024). The Art of Cybercrime Community Research. ACM Comput. Surv. 56, 6, Article 155 (June 2024), 26 pages. DOI:10.1145/3639362 (дата звернення: 14. 10. 2024).

3. Gnatyuk Sergiy. Система корелювання подій та управління інцидентами кібербезпеки на об’єктах критичної інфраструктури. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка», 2023. 3.19: 181 с.

4. Каплунов А. В., засоби моніторингу мережі в іот інфраструктурі з гібридною архітектурою. Телекомунікаційні та інформаційні технології, 2023, 2: 29 с.

5. Nagios XI URL: https://www.usenix.org/system/files/login/articles/10josephsen_069-073_online.pdf (дата звернення: 14. 10. 2024).

6. Nagios XI // Use Cases & Benefits URL: https://assets.nagios.com/handouts/nagiosxi/Nagios-XI-Use-Cases-and-Benefits.pdf (дата звернення: 14. 10. 2024).

7. ManageEngine OpManager URL: https://www.manageengine.com/network-monitoring/ (дата звернення: 14. 10. 2024).

8. OpManager 5 Benefits - Differential management URL: https://www.manageengine.com/network-monitoring/benefits.html (дата звернення: 14. 10. 2024).

9. Paessler PRTG URL: https://www.paessler.com/prtg (дата звернення: 14. 10. 2024).

10. Pros and Cons of PRTG 2024 URL: https://www.trustradius.com/products/prtg-network-monitor/reviews?qs=pros-and-cons (дата звернення: 14. 10. 2024).

11. Zabbix 4.2. URL: https://blog.zabbix.com/zabbix-4-2-out-now/6791/ (дата звернення 14.10.2024).

12. Keep control of your infrastructure by collecting any metric from any source. URL: https://www.zabbix.com/features (дата звернення 14.10.2024).

13. The Best Network Monitoring Software and Tools of 2024. URL: https://www.comparitech.com/net-admin/network-monitoring-tools/ (дата звернення 14.10.2024).

14. Zabbix Ratings Overview. URL: https://www.gartner.com/reviews/market/infrastructure-monitoring-tools/vendor/zabbix/product/zabbix (дата звернення 14.10.2024).

15. Seifert Jeffrey W. Data mining: An overview. National security issues, 2004. 201 p.

16. Dubrovin Valerii; Deineha Larysa; Yatsenko Anastasiya. Програмне забезпечення статистичного аналізу. Electrical Engineering and Power Engineering, 2023. 27 p.

17. Гороховатський В. О. Методи інтелектуального аналізу та оброблення даних : навч. посібник; М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків : ХНУРЕ, 2021. – 92 с.

18. Ланде Д.В., Субач І.Ю., Бояринова Ю.Є. Основи теорії і практики інтелектуального аналізу даних у сфері кібербезпеки: навч. посібник. Київ:ІСЗЗІ КПІ ім. Ігоря Сікорського, 2018. –297 с.

19. Башкатов Є. О. Застосування методів машинного навчання при обробці великих даних М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків, 2022. – 52 с.

20. Наквасюк Василь. Дерева рішень і алгоритми їх побудови. Київ. 2020. – 8с.

21. Базилевич В. П.; Подольський І. В.; Базилевич П. Р. Ієрархічна кластеризація–ефективний засіб розв'язування неполіноміальних комбінаторних задач схемного типу. Штучний інтелект.-НАН України, 2002, №3, 483с.

22. Nadkarni Prakash M.; Ohno-Machado Lucila; Chapman Wendy W. Natural language processing: an introduction. Journal of the American Medical Informatics Association, 2011. Vol 18, No5, 546 p.

23. KANG Yue. Natural language processing (NLP) in management research: A literature review. Journal of Management Analytics, 2020. 172 p.

24. Alzubaidi Laith. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. Journal of big Data, 2021. Vol. 8, pp. 1-74 p.

25. Кравченко С. М.; Гришкун Є. О.; Власенко О. В. Методи класифікації машинного навчання з використанням бібліотеки scikit-learn. Вчені записки Таврійського національного університету імені ВІ Вернадського. Серія: Технічні науки, 2020. 124с.

26. Jakkula Vikramaditya. Tutorial on support vector machine (svm). School of EECS, Washington State University, 2006. Vol. 2, No 5, 3 p.

27. Моторна Я., Красношлик Н; Піскун О. Реалізація та дослідження алгоритму random forest для розв'язування задач класифікації. Cherkasy University Bulletin: Applied Mathematics. Informatics, 2020. 1 p.

28. Бабіч О., Дух Д., Глухова А. Особливості методів машинного навчання щодо їх використання в процесі аналізу англомовних джерел. Збірник наукових праць Військового інституту Київського національного університету імені Тараса Шевченка, 2018, 61: 34 с.

29. Крамський С.; Захарченко О. Організаційна модель управління етапами реалізації програм інфраструктурних проєктів. Управління розвитком складних систем, 2022, 52: 29 с.

30. Zargar Saman Taghavi; Joshi James; Tipper David. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. IEEE communications surveys & tutorials, 2013. Vol 15, No4, 2052 p.

31. Балабанов О.І., Павленко А.П. PyCharm для розробників Python: Навчальний посібник. - Львів: Видавництво Львівської політехніки, 2018. - 300 с.

32. Гальперін А., Кочнев Д. Робота з Eclipse: Навчальний посібник. - Харків: Видавництво Харківського національного університету імені В.Н. Каразіна, 2017. - 230 с.

33. 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.

34. Козловська В. Організація баз даних та знань: конспект лекцій. Одеса, Одеський державний екологічний університет, 2019. – 130с.

35. Nixon R. Learning PHP, MySQL, JavaScript, and CSS: A step-by-step guide to creating dynamic websites. – «O'Reilly Media, Inc.», 2012. – 315 p.

36. Іван Петров. Основи Firebird SQL. – Харків, видавництво "Фоліо" 2020. – 350 с. Транслітерація: Ivan Petrov. Fundamentals of Firebird SQL. / Packt Publishing – Birmingham, 2018. – 350 p.

37. Lee, Y., Scolari, A., Chun, B. G., Weimer, M., & Interlandi, M. From the Edge to the Cloud: Model Serving in ML. NET. IEEE Data Eng. Bull. 2018. Vol. 41, No 4, pp. 46-53.

38. ML.NET: Machine Learning for .NET Developers. URL: https://www.codemag.com/Article/1911042/ML.NET-Machine-Learning-for-.NET-Developers (дата звернення 05.10.2024).

39. Thatavarthi, N. L. S. Developing AI and ML Solutions with ML. Net. Journal of Artificial Intelligence & Cloud Computing. 2022. Vol. 358, pp. 2-3.

40. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Zheng, X. {TensorFlow}: a system for {Large-Scale} machine learning. In 12th USENIX symposium on operating systems design and implementation. 2016. pp. 265-283.

41. Das, R. S. TensorFlow: Revolutionizing Large-Scale Machine Learning in Complex Semiconductor Design. International Journal of Computing and Engineering. 2024. Vol. 5, No 3, pp. 1-9.

42. Singh, P., Manure, A., Singh, P., & Manure, A. Introduction to tensorflow 2.0. Learn TensorFlow 2.0: Implement Machine Learning and Deep Learning Models with Python. 2020. pp. 1-24.

43. Kochnev, A. Using neural networks in the problem of classifying anomalous behavior in financial transactions using Python and Keras. International Journal of Open Information Technologies. 2023. Vol. 11, No 10, pp. 55-61.

44. Basics of Keras Layers. URL: https://techvidvan.com/tutorials/keras-layers/ (дата звернення 05.10.2024).

45. Shobha, Y., Prasad, K. V., Anuradha, S. G., & Vaidya, H. Auto Skin Tumour Classification Using CNN Framework with Tensorflow and Keras. Mathematical Statistician and Engineering Applications. 2023. Vol. 72, No 1, pp. 172-184.

46. Cyber Attacks on Real-Time Internet of Things . URL: https://www.kaggle.com/datasets/joebeachcapital/real-time-internet-of-things-rt-iot2022 (дата звернення 15.11.2024).

Додаток А. Лістинги програмного коду

Лістинг 1. Код класу «PredictorForm»

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

namespace InfrastructureMonitoringApp.Forms.Controls {
  public partial class PredictorForm : Form {
    private ValidationMy _Validation = new ValidationMy();
    private Models _SelectedModels = new Models();

    private MLContext _Context = new MLContext();
    private PredictionEngine<InputData, Prediction> predictionEngine;
    private ModelsProvider _ModelsProvider = new ModelsProvider();
    private List<Models> _ModelsL = new List<Models>();



    private bool _IsModelsLoad = false;
    private List<InputData> _InputData = new List<InputData>();
    readonly string _testDataPath = Path.Combine(Environment.CurrentDirectory, "Data",
"test_data.csv");

    public PredictorForm() {
      InitializeComponent();
      LoadAllDate();
    }

    private void LoadAllDate() {
      ProtoCBox.SelectedIndex = 0;
      ServiceCBox.SelectedIndex = 0;
      _ModelsL = _ModelsProvider.GetAllModels();
      ModelsCBox.DataSource = _ModelsL;
      ModelsCBox.ValueMember = "ModelsId";
      ModelsCBox.DisplayMember = "ModelsNames";
      _IsModelsLoad = true;
```

```
      _SelectedModels =
_ModelsProvider.SelectedModelsByModelsId(Convert.ToInt32(ModelsCBox.SelectedValue));
      LoadData(_SelectedModels.ModelsFileModel);
      LoadTestDataFromFile();

    }


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


    private void LoadData(string FilePath) {
      string localProj = Application.StartupPath + FilePath;
      // Define DataViewSchema for data preparation pipeline and trained model
      DataViewSchema modelSchema;
      // Load trained model
      ITransformer model = _Context.Model.Load(localProj, out modelSchema);
      // Evaluate the model
      // Use the model to make predictions
      predictionEngine = _Context.Model.CreatePredictionEngine<InputData, Prediction>(model);
    }

    private void GenBtn_Click(object sender, EventArgs e) {
      if (IsModelsCorrect()) {
        if (timer1.Enabled) {
          timer1.Enabled = false;
          GenBtn.Text = "Моніторити";
        } else {
          timer1.Enabled = true;
          GenBtn.Text = "Зупинити";
        }
      }
    }

    private void timer1_Tick(object sender, EventArgs e) {
      PredictAndDisplay();
    }

    private void PredictAndDisplay() {
      // Перевірка наявності даних у списку
      if (_InputData.Count == 0) {
        RaportTBox.Text = "Список даних порожній.";
        return;
      }

      // Вибір випадкового запису зі списку
```

```
    var random = new Random();
    var randomInputData = _InputData[random.Next(_InputData.Count)];

    // Новий екземпляр класу InputData для прогнозування
    var inputDataForPrediction = new InputData {
      id = randomInputData.id,
      id_orig_p = randomInputData.id_orig_p,
      id_resp_p = randomInputData.id_resp_p,
      proto = randomInputData.proto,
      service = randomInputData.service,
      flow_duration = randomInputData.flow_duration,
      fwd_pkts_tot = randomInputData.fwd_pkts_tot,
      bwd_pkts_tot = randomInputData.bwd_pkts_tot,
      fwd_data_pkts_tot = randomInputData.fwd_data_pkts_tot,
      bwd_data_pkts_tot = randomInputData.bwd_data_pkts_tot,
      fwd_pkts_per_sec = randomInputData.fwd_pkts_per_sec,
      bwd_pkts_per_sec = randomInputData.bwd_pkts_per_sec,
      flow_pkts_per_sec = randomInputData.flow_pkts_per_sec,
      down_up_ratio = randomInputData.down_up_ratio,
      fwd_header_size_tot = randomInputData.fwd_header_size_tot,
      fwd_header_size_min = randomInputData.fwd_header_size_min,
      fwd_header_size_max = randomInputData.fwd_header_size_max,
      bwd_header_size_tot = randomInputData.bwd_header_size_tot,
      bwd_header_size_min = randomInputData.bwd_header_size_min,
      bwd_header_size_max = randomInputData.bwd_header_size_max,
      flow_FIN_flag_count = randomInputData.flow_FIN_flag_count
    };

    // Прогноз за допомогою вже створеного predictionEngine
    var prediction = predictionEngine.Predict(inputDataForPrediction);

    // Виведення повної інформації про всі значення полів
    var info = new StringBuilder();
    info.AppendLine("--- Дані для прогнозування ---");
    info.AppendLine($"ID: {randomInputData.id}");
    info.AppendLine($"ID оригінального джерела: {randomInputData.id_orig_p}");
    info.AppendLine($"ID респондента: {randomInputData.id_resp_p}");
    info.AppendLine($"Протокол: {randomInputData.proto}");
    info.AppendLine($"Служба: {randomInputData.service}");
    info.AppendLine($"Тривалість потоку: {randomInputData.flow_duration} мс");
    info.AppendLine($"Загальна кількість всіх пакетів (від джерела):
{randomInputData.fwd_pkts_tot}");
    info.AppendLine($"Загальна кількість всіх пакетів (від одержувача):
{randomInputData.bwd_pkts_tot}");
    info.AppendLine($"Кількість пакетів, що містять лише дані (від джерела):
{randomInputData.fwd_data_pkts_tot}");
    info.AppendLine($"Кількість пакетів, що містять лише дані (від одержувача):
{randomInputData.bwd_data_pkts_tot}");
    info.AppendLine($"Кількість пакетів, що надходять за секунду: (від джерела):
{randomInputData.fwd_pkts_per_sec}");
    info.AppendLine($" Кількість пакетів, що відправляються  за секунду (від одержувача):
{randomInputData.bwd_pkts_per_sec}");
```

```
    info.AppendLine($"Кількість пакетів у потоці за секунду:
{randomInputData.flow_pkts_per_sec}");
    info.AppendLine($"Відношення пакетів вниз/вгору: {randomInputData.down_up_ratio}");
    info.AppendLine($"Загальний розмір заголовків (від джерела):
{randomInputData.fwd_header_size_tot}");
    info.AppendLine($"Мінімальний розмір заголовка (від джерела):
{randomInputData.fwd_header_size_min}");
    info.AppendLine($"Максимальний розмір заголовка (від джерела):
{randomInputData.fwd_header_size_max}");
    info.AppendLine($"Загальний розмір заголовків (від одержувача):
{randomInputData.bwd_header_size_tot}");
    info.AppendLine($"Мінімальний розмір заголовка (від одержувача):
{randomInputData.bwd_header_size_min}");
    info.AppendLine($"Максимальний розмір заголовка (від одержувача):
{randomInputData.bwd_header_size_max}");
    info.AppendLine($"Кількість встановлених прапорців FIN у потоці:
{randomInputData.flow_FIN_flag_count}");

    // Виведення прогнозу
    info.AppendLine("\r\n--- Прогноз ---");
    info.AppendLine($"Передбачений тип атаки: {prediction.PredictedLabel}");
    info.AppendLine($"Оцінки для кожного класу: {string.Join(", ", prediction.Score)}");

    // Виведення результату у текстове поле
    RaportTBox.Text = info.ToString();
  }


  private void PredictBtn_Click(object sender, EventArgs e) {
    if (IsAllInputDataCorrect()) {
      // Створення нового екземпляра InputData з введених користувачем даних
      var inputData = new InputData {
        id = (float)Convert.ToDouble(IdTBox.Text),
        id_orig_p = (float)Convert.ToDouble(Id_Orig_PTBox.Text),
        id_resp_p = (float)Convert.ToDouble(Id_Resp_PTBox.Text),
        proto = ProtoCBox.Text,
        service = ServiceCBox.Text,
        flow_duration = (float)Convert.ToDouble(Flow_DurationTBox.Text),
        fwd_pkts_tot = (float)Convert.ToDouble(Fwd_Pkts_TotTBox.Text),
        bwd_pkts_tot = (float)Convert.ToDouble(Bwd_Pkts_TotTBox.Text),
        fwd_data_pkts_tot = (float)Convert.ToDouble(Fwd_Data_Pkts_TotTBox.Text),
        bwd_data_pkts_tot = (float)Convert.ToDouble(Bwd_Data_Pkts_TotTBox.Text),
        fwd_pkts_per_sec = (float)Convert.ToDouble(Fwd_Pkts_Per_SecTBox.Text),
        bwd_pkts_per_sec = (float)Convert.ToDouble(Bwd_Pkts_Per_SecTBox.Text),
        flow_pkts_per_sec = (float)Convert.ToDouble(Flow_Pkts_Per_SecTBox.Text),
        down_up_ratio = (float)Convert.ToDouble(Down_Up_RatioTBox.Text),
        fwd_header_size_tot = (float)Convert.ToDouble(Fwd_Header_Size_TotTBox.Text),
        fwd_header_size_min = (float)Convert.ToDouble(Fwd_Header_Size_MinTBox.Text),
        fwd_header_size_max = (float)Convert.ToDouble(Fwd_Header_Size_MaxTBox.Text),
        bwd_header_size_tot = (float)Convert.ToDouble(Bwd_Header_Size_TotTBox.Text),
        bwd_header_size_min = (float)Convert.ToDouble(Bwd_Header_Size_MinTBox.Text),
        bwd_header_size_max = (float)Convert.ToDouble(Bwd_Header_Size_MaxTBox.Text),
```

```
        flow_FIN_flag_count = (float)Convert.ToDouble(Flow_FIN_Flag_CountTBox.Text)
    };

    // Виведення інформації про введені дані
    var info = new StringBuilder();
    info.AppendLine("--- Дані для прогнозування ---");
    info.AppendLine($"ID: {inputData.id}");
    info.AppendLine($"ID оригінального джерела: {inputData.id_orig_p}");
    info.AppendLine($"ID респондента: {inputData.id_resp_p}");
    info.AppendLine($"Протокол: {inputData.proto}");
    info.AppendLine($"Служба: {inputData.service}");
    info.AppendLine($"Тривалість потоку: {inputData.flow_duration} мс");
    info.AppendLine($"Загальна кількість всіх пакетів (від джерела): {inputData.fwd_pkts_tot}");
    info.AppendLine($"Загальна кількість всіх пакетів (від одержувача):
{inputData.bwd_pkts_tot}");
    info.AppendLine($"Кількість пакетів, що містять лише дані (від джерела):
{inputData.fwd_data_pkts_tot}");
    info.AppendLine($"Кількість пакетів, що містять лише дані (від одержувача):
{inputData.bwd_data_pkts_tot}");
    info.AppendLine($"Кількість пакетів, що надходять за секунду (від джерела):
{inputData.fwd_pkts_per_sec}");
    info.AppendLine($"Кількість пакетів, що відправляються за секунду (від одержувача):
{inputData.bwd_pkts_per_sec}");
    info.AppendLine($"Кількість пакетів у потоці за секунду: {inputData.flow_pkts_per_sec}");
    info.AppendLine($"Відношення пакетів вниз/вгору: {inputData.down_up_ratio}");
    info.AppendLine($"Загальний розмір заголовків (від джерела):
{inputData.fwd_header_size_tot}");
    info.AppendLine($"Мінімальний розмір заголовка (від джерела):
{inputData.fwd_header_size_min}");
    info.AppendLine($"Максимальний розмір заголовка (від джерела):
{inputData.fwd_header_size_max}");
    info.AppendLine($"Загальний розмір заголовків (від одержувача):
{inputData.bwd_header_size_tot}");
    info.AppendLine($"Мінімальний розмір заголовка (від одержувача):
{inputData.bwd_header_size_min}");
    info.AppendLine($"Максимальний розмір заголовка (від одержувача):
{inputData.bwd_header_size_max}");
    info.AppendLine($"Кількість встановлених прапорців FIN у потоці:
{inputData.flow_FIN_flag_count}");

    // Передбачення на основі введених даних
    var prediction = predictionEngine.Predict(inputData);

    // Отримання передбаченого типу атаки та рекомендацій
    var attackType = prediction.PredictedLabel;
    var scores = string.Join(", ", prediction.Score);

    // Формування виведення результату прогнозу
    info.AppendLine("\r\n--- Прогноз ---");
    info.AppendLine($"Передбачений тип атаки: {attackType}");
    info.AppendLine($"Оцінки для кожного класу: {scores}");
```

```
    // Виведення результату у текстове поле
    RaportTBox.Text = info.ToString();
  }
}



private void LoadTestDataFromFile() {
  // Перевіряємо, чи існує файл із тестовими даними
  if (!File.Exists(_testDataPath)) {
    RaportTBox.Text = "Файл з тестовими даними не знайдено.";
    return;
  }

  // Читаємо CSV-файл і заповнюємо список _InputData
  using (var reader = new StreamReader(_testDataPath)) {
    string line;
    bool isFirstLine = true;

    while ((line = reader.ReadLine()) != null) {
      // Пропускаємо перший рядок (заголовки)
      if (isFirstLine) {
        isFirstLine = false;
        continue;
      }

      // Розділяємо рядок CSV на стовпці
      var values = line.Split(',');

      // Перевіряємо, чи правильна кількість стовпців (має бути 21, не враховуючи Attack_type)
      if (values.Length >= 21) {
        var inputData = new InputData {
          id = float.Parse(values[0], CultureInfo.InvariantCulture),
          id_orig_p = float.Parse(values[1], CultureInfo.InvariantCulture),
          id_resp_p = float.Parse(values[2], CultureInfo.InvariantCulture),
          proto = values[3],
          service = values[4],
          flow_duration = float.Parse(values[5], CultureInfo.InvariantCulture),
          fwd_pkts_tot = float.Parse(values[6], CultureInfo.InvariantCulture),
          bwd_pkts_tot = float.Parse(values[7], CultureInfo.InvariantCulture),
          fwd_data_pkts_tot = float.Parse(values[8], CultureInfo.InvariantCulture),
          bwd_data_pkts_tot = float.Parse(values[9], CultureInfo.InvariantCulture),
          fwd_pkts_per_sec = float.Parse(values[10], CultureInfo.InvariantCulture),
          bwd_pkts_per_sec = float.Parse(values[11], CultureInfo.InvariantCulture),
          flow_pkts_per_sec = float.Parse(values[12], CultureInfo.InvariantCulture),
          down_up_ratio = float.Parse(values[13], CultureInfo.InvariantCulture),
          fwd_header_size_tot = float.Parse(values[14], CultureInfo.InvariantCulture),
          fwd_header_size_min = float.Parse(values[15], CultureInfo.InvariantCulture),
          fwd_header_size_max = float.Parse(values[16], CultureInfo.InvariantCulture),
          bwd_header_size_tot = float.Parse(values[17], CultureInfo.InvariantCulture),
          bwd_header_size_min = float.Parse(values[18], CultureInfo.InvariantCulture),
          bwd_header_size_max = float.Parse(values[19], CultureInfo.InvariantCulture),
```

```
            flow_FIN_flag_count = float.Parse(values[20], CultureInfo.InvariantCulture)
          };

          _InputData.Add(inputData);
        }
      }
    }

    // Виводимо повідомлення, що дані успішно завантажені
    RaportTBox.Text = "Тестові дані успішно завантажені.";
}


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

private bool IsAllInputDataCorrect() {
    bool isCorrect = true;
    if (_Validation.IsDataConvertToInt(IdTBox.Text)) {
        IdValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
        IdValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(Id_Orig_PTBox.Text)) {
        Id_Orig_PValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
        Id_Orig_PValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(Id_Resp_PTBox.Text)) {
        Id_Resp_PValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
        Id_Resp_PValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(Flow_DurationTBox.Text)) {
        Flow_DurationValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
        Flow_DurationValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(Fwd_Pkts_TotTBox.Text)) {
        Fwd_Pkts_TotValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
```

```
} else {
  Fwd_Pkts_TotValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(Bwd_Pkts_TotTBox.Text)) {
  Bwd_Pkts_TotValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  Bwd_Pkts_TotValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(Fwd_Data_Pkts_TotTBox.Text)) {
  Fwd_Data_Pkts_TotValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  Fwd_Data_Pkts_TotValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(Bwd_Data_Pkts_TotTBox.Text)) {
  Bwd_Data_Pkts_TotValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  Bwd_Data_Pkts_TotValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(Fwd_Pkts_Per_SecTBox.Text)) {
  Fwd_Pkts_Per_SecValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  Fwd_Pkts_Per_SecValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(Bwd_Pkts_Per_SecTBox.Text)) {
  Bwd_Pkts_Per_SecValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  Bwd_Pkts_Per_SecValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(Flow_Pkts_Per_SecTBox.Text)) {
  Flow_Pkts_Per_SecValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  Flow_Pkts_Per_SecValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(Down_Up_RatioTBox.Text)) {
  Down_Up_RatioValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  Down_Up_RatioValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
if (_Validation.IsDataConvertToDouble(Fwd_Header_Size_TotTBox.Text)) {
  Fwd_Header_Size_TotValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
} else {
  Fwd_Header_Size_TotValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
  isCorrect = false;
}
```

```
    if (_Validation.IsDataConvertToDouble(Fwd_Header_Size_MinTBox.Text)) {
      Fwd_Header_Size_MinValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      Fwd_Header_Size_MinValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(Fwd_Header_Size_MaxTBox.Text)) {
      Fwd_Header_Size_MaxValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      Fwd_Header_Size_MaxValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(Bwd_Header_Size_TotTBox.Text)) {
      Bwd_Header_Size_TotValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      Bwd_Header_Size_TotValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(Bwd_Header_Size_MinTBox.Text)) {
      Bwd_Header_Size_MinValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      Bwd_Header_Size_MinValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(Bwd_Header_Size_MaxTBox.Text)) {
      Bwd_Header_Size_MaxValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      Bwd_Header_Size_MaxValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_Validation.IsDataConvertToDouble(Flow_FIN_Flag_CountTBox.Text)) {
      Flow_FIN_Flag_CountValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      Flow_FIN_Flag_CountValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    return isCorrect;
  }


 }
}
```

Лістинг 2. Код класу «ModelsForm»

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

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

namespace InfrastructureMonitoringApp.Forms.Dictinary {
  public partial class ModelsForm : Form {
    private MLContext mlContext;
    private ITransformer model;
    private IDataView dataView;

    private string _Path = "";

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

    private async void OpenBtn_Click(object sender, EventArgs e) {
      // Створення діалогового вікна для відкриття файлу
      OpenFileDialog openFileDialog = new OpenFileDialog();

      // Налаштування властивостей діалогового вікна
      openFileDialog.Filter = "Text files (*.csv)|*.csv|All files (*.*)|*.*";
      openFileDialog.FilterIndex = 2;
      openFileDialog.RestoreDirectory = true;

      // Відображення діалогового вікна та обробка результату
      if (openFileDialog.ShowDialog() == DialogResult.OK) {
        _Path = openFileDialog.FileName;
        FileNameTBox.Text = openFileDialog.FileName;

        // Створення контексту ML
        mlContext = new MLContext(seed: 0);

        // Завантаження даних
        dataView = mlContext.Data.LoadFromTextFile<InputData>(_Path,
            hasHeader: true,
            separatorChar: ',');

        // Виведення повідомлення про тренування моделі
        RaportTBox.Text = "Тренування моделі ..." + "\r\n";

        // Виведення кількості записів по кожному типу атаки
```

```
RaportTBox.Text += ("==== Кількість записів по кожному типу атаки ===\r\n");

var attackTypeCounts =
  mlContext.Data.CreateEnumerable<InputData>(dataView, reuseRowObject: false)
    .GroupBy(x => x.Attack_type)
    .Select(g => new { AttackType = g.Key, Count = g.Count() })
    .OrderByDescending(g => g.Count);

foreach (var item in attackTypeCounts) {
  RaportTBox.Text += ($"{item.AttackType} - {item.Count}\r\n");
}

// Розділення даних на тренувальний та тестовий набори
var trainTestSplit = mlContext.Data.TrainTestSplit(dataView, testFraction: 0.2);

//Нормалізація даних і попередня обробка
var dataProcessPipeline = mlContext.Transforms.Categorical.OneHotEncoding(
    outputColumnName: "protoEncoded", inputColumnName: "proto")
.Append(mlContext.Transforms.Categorical.OneHotEncoding(
    outputColumnName: "serviceEncoded", inputColumnName: "service"))
.Append(mlContext.Transforms.Concatenate("Features",
    // Числові ознаки
    nameof(InputData.id), nameof(InputData.id_orig_p),
    nameof(InputData.id_resp_p), nameof(InputData.flow_duration),
    nameof(InputData.fwd_pkts_tot), nameof(InputData.bwd_pkts_tot),
    nameof(InputData.fwd_data_pkts_tot), nameof(InputData.bwd_data_pkts_tot),
    nameof(InputData.fwd_pkts_per_sec),  nameof(InputData.bwd_pkts_per_sec),
    nameof(InputData.flow_pkts_per_sec), nameof(InputData.down_up_ratio),
    nameof(InputData.fwd_header_size_tot), nameof(InputData.fwd_header_size_min),
    nameof(InputData.fwd_header_size_max), nameof(InputData.bwd_header_size_tot),
    nameof(InputData.bwd_header_size_min), nameof(InputData.bwd_header_size_max),
    nameof(InputData.flow_FIN_flag_count),
    "protoEncoded",
    "serviceEncoded"))
.Append(mlContext.Transforms.ReplaceMissingValues(
    inputColumnName: "Features", outputColumnName: "Features"))
.Append(mlContext.Transforms.NormalizeMinMax("Features"))
.Append(mlContext.Transforms.Conversion.MapValueToKey("Label", "Attack_type"));

//Вибір тренера
var trainer =
  mlContext.MulticlassClassification.Trainers.LbfgsMaximumEntropy("Label", "Features")
    .Append(mlContext.Transforms.Conversion.MapKeyToValue("PredictedLabel"));
var trainingPipeline = dataProcessPipeline.Append(trainer);
// Навчання моделі
model = trainingPipeline.Fit(trainTestSplit.TrainSet);

// Оцінка моделі
RaportTBox.Text += ("=============== Оцінка моделі ===============\r\n");

// Отримуємо перетворений тестовий набір
var predictions = model.Transform(trainTestSplit.TestSet);
```

```
    // Оцінюємо модель
    var metrics = mlContext.MulticlassClassification.Evaluate(predictions);
    // Заголовок для метрик
    RaportTBox.Text += ("Метрики для кожного типу вразливостей:\r\n");
    // Отримуємо матрицю помилок
    var confusionMatrix = metrics.ConfusionMatrix;
    // Отримуємо кількість класів
    int classCount = confusionMatrix.NumberOfClasses;

    // Отримуємо назви класів з метаданих стовпця 'Score'
    VBuffer<ReadOnlyMemory<char>> slotNames = default;
    predictions.Schema["Score"].GetSlotNames(ref slotNames);
    var classNames = slotNames.DenseValues().Select(x => x.ToString()).ToArray();
    // Отримуємо матрицю помилок як 2D-масив
    var counts = confusionMatrix.Counts;
    // Масиви для зберігання метрик
    double[] perClassPrecision = new double[classCount];
    double[] perClassRecall = new double[classCount];
    double[] perClassF1Score = new double[classCount];

    // Цикл для обчислення метрик по кожному класу
    for (int i = 0; i < classCount; i++) {
      double truePositive = counts[i][i];
      double falsePositive = 0;
      double falseNegative = 0;
      // Обчислюємо False Positive та False Negative для кожного класу
      for (int j = 0; j < classCount; j++) {
        if (i != j) {
          falsePositive += counts[j][i];
          falseNegative += counts[i][j];
        }
      }
      // Обчислюємо Precision, Recall та F1 Score
      perClassPrecision[i] = truePositive / (truePositive + falsePositive + 1e-6);
      perClassRecall[i] = truePositive / (truePositive + falseNegative + 1e-6);
      perClassF1Score[i] = 2 * perClassPrecision[i] * perClassRecall[i] /
                           (perClassPrecision[i] + perClassRecall[i] + 1e-6);
      // Виводимо метрики для кожного класу
      string className = classNames.Length > i ? classNames[i] : $"Клас {i}";
      RaportTBox.Text += ($"Тип вразливості: {className}\r\n");
      RaportTBox.Text += ($"  Precision: {perClassPrecision[i]:P2}\r\n");
      RaportTBox.Text += ($"  Recall: {perClassRecall[i]:P2}\r\n");
      RaportTBox.Text += ($"  F1 Score: {perClassF1Score[i]:P2}\r\n");
    }

    _IsModelTrain = true;
    SetVisibleData();
  }
}

// Функція для обчислення суми всіх елементів у IReadOnlyList<IReadOnlyList<double>>
double CalculateTotalSum(IReadOnlyList<IReadOnlyList<double>> counts) {
```

```
      double totalSum = 0;
      foreach (var row in counts) {
        totalSum += row.Sum();
      }
      return totalSum;
    }

    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        //Save model
        string pathName = @"\teach\" + GenerateFileName() + ".zip";
        string localProj =

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

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

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

    private void SetVisibleData() {
      // Встановлюємо позицію курсора на останній символ у TextBox
      RaportTBox.SelectionStart = RaportTBox.Text.Length;
      // Прокручуємо до позиції курсора
      RaportTBox.ScrollToCaret();
    }

    private void DataLoad() {
      int firstRowIndex = 0;
      if (ModelsGridView.FirstDisplayedScrollingRowIndex > 0) {
        firstRowIndex = ModelsGridView.FirstDisplayedScrollingRowIndex;
      }
      try {
        _ModelsList = _ModelsProvider.GetAllModels();
        LoadDataInModelsGridView(_ModelsList);
        if (_selectedRowIndex == ModelsGridView.Rows.Count) {
          _selectedRowIndex = ModelsGridView.Rows.Count - 1;
        }
        if (_selectedRowIndex >= 0) {
          ModelsGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
```

```
      ModelsGridView.Rows[_selectedRowIndex].Selected = true;
    }
  } catch (Exception ex) {
    MessageBox.Show(ex.ToString());
  }
}

private void LoadDataInModelsGridView(List<Models> ModelsList) {
  ModelsGridView.DataSource = null;
  ModelsGridView.Columns.Clear();
  ModelsGridView.AutoGenerateColumns = false;
  ModelsGridView.RowHeadersVisible = false;

  ModelsGridView.DataSource = ModelsList;

  if (ModelsList.Count > 0) {
    if (ModelsList[0].Message == NamesMy.NoDataNames.NoDataInModels) {
      DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
      messageColumn.DataPropertyName = "Message";
      messageColumn.Width = ModelsGridView.Width - NamesMy.SizeOptins.MinusSizePanel;
      ModelsGridView.Columns.Add(messageColumn);
    } else {
      DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
      DetailIdColumn.DataPropertyName = "ModelsId";
      ModelsGridView.Columns.Add(DetailIdColumn);
      ModelsGridView.Columns[0].Visible = false;

      DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
      numberColumn.HeaderText = "№ ";
      numberColumn.DataPropertyName = "Number";
      numberColumn.DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleRight;
      numberColumn.Width = NamesMy.SizeOptins.NumberSize;
      ModelsGridView.Columns.Add(numberColumn);

      DataGridViewColumn ModelsNamesColumn = new DataGridViewTextBoxColumn();
      ModelsNamesColumn.HeaderText = "Назва моделі";
      ModelsNamesColumn.DataPropertyName = "ModelsNames";
      ModelsNamesColumn.Width = 200;
      ModelsGridView.Columns.Add(ModelsNamesColumn);

      DataGridViewColumn TrainDateColumn = new DataGridViewTextBoxColumn();
      TrainDateColumn.HeaderText = "Дата/час тренування";
      TrainDateColumn.DataPropertyName = "TrainDate";
      TrainDateColumn.DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleRight;
      TrainDateColumn.Width = NamesMy.SizeOptins.NumberSize;
      ModelsGridView.Columns.Add(TrainDateColumn);

      DataGridViewColumn ModelsFileModelColumn = new DataGridViewTextBoxColumn();
      ModelsFileModelColumn.HeaderText = "Файл";
      ModelsFileModelColumn.DataPropertyName = "ModelsFileModel";
      ModelsFileModelColumn.Width = 200;
      ModelsGridView.Columns.Add(ModelsFileModelColumn);
```

```
        DataGridViewButtonColumn IsResidesBtn = new DataGridViewButtonColumn();
        IsResidesBtn.HeaderText = "Видалити";
        IsResidesBtn.Text = "Видалити";
        IsResidesBtn.UseColumnTextForButtonValue = true;
        IsResidesBtn.ToolTipText = "Видалити";
        IsResidesBtn.Width = NamesMy.SizeOptins.DeleteBtnSize;
        ModelsGridView.Columns.Add(IsResidesBtn);

      }
      for (int i = 0; i < ModelsGridView.Columns.Count; i++) {
        ModelsGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
      }
    }
  }

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

    return fileName;
  }

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

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

```
        DataLoad();
       }
      }
     }

  }
}
```

Лістинг 3. Код класу «DataModelsProvider»

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

namespace InfrastructureMonitoringApp.Providers {
  internal class DataModelsProvider {

  }
}

// Клас вхідних даних для обробки мережевого трафіку
public class InputData {
  // Ідентифікатор потоку або запису трафіку
  [LoadColumn(0)]
  public float id;

  // Ідентифікатор оригінального джерела (адреса або інша інформація про джерело)
  [LoadColumn(1)]
  public float id_orig_p;

  // Ідентифікатор цільового (респонсивного) кінцевого вузла в мережі
  [LoadColumn(2)]
  public float id_resp_p;

  // Протокол, який використовується в цьому потоці (наприклад, TCP, UDP)
  [LoadColumn(3)]
  public string proto;

  // Тип служби, що використовується в потоці (наприклад, HTTP, DNS)
  [LoadColumn(4)]
  public string service;

  // Тривалість потоку в мілісекундах
  [LoadColumn(5)]
  public float flow_duration;

  // Загальна кількість всіх пакетів, переданих від джерела (вперед) за потік
  [LoadColumn(6)]
  public float fwd_pkts_tot;
```

```
// Загальна кількість всіх пакетів, отриманих від одержувача (назад) за потік
[LoadColumn(7)]
public float bwd_pkts_tot;

// Кількість пакетів, що містять лише дані, передані від джерела (вперед)
[LoadColumn(8)]
public float fwd_data_pkts_tot;

// Кількість пакетів, що містять лише дані, отримані від одержувача (назад)
[LoadColumn(9)]
public float bwd_data_pkts_tot;

// Кількість пакетів, що відправляються вперед за секунду
[LoadColumn(10)]
public float fwd_pkts_per_sec;

// Кількість пакетів, що надходять назад за секунду
[LoadColumn(11)]
public float bwd_pkts_per_sec;

// Кількість пакетів у потоці за секунду
[LoadColumn(12)]
public float flow_pkts_per_sec;

// Відношення пакетів вниз до пакетів вгору
[LoadColumn(13)]
public float down_up_ratio;

// Загальний розмір заголовків, надісланих вперед
[LoadColumn(14)]
public float fwd_header_size_tot;

// Мінімальний розмір заголовка, надісланого вперед
[LoadColumn(15)]
public float fwd_header_size_min;

// Максимальний розмір заголовка, надісланого вперед
[LoadColumn(16)]
public float fwd_header_size_max;

// Загальний розмір заголовків, надісланих назад
[LoadColumn(17)]
public float bwd_header_size_tot;

// Мінімальний розмір заголовка, надісланого назад
[LoadColumn(18)]
public float bwd_header_size_min;

// Максимальний розмір заголовка, надісланого назад
[LoadColumn(19)]
public float bwd_header_size_max;
```

```
  // Кількість встановлених прапорців FIN у потоці (завершення TCP-з'єднання)
  [LoadColumn(20)]
  public float flow_FIN_flag_count;

  // Тип атаки, який ідентифікується в цьому записі (мітка класу)
  [LoadColumn(84)]
  [ColumnName("Attack_type")]
  public string Attack_type;
}


// Клас прогнозування для моделі машинного навчання
public class Prediction {
  // Передбачена моделью мітка класу (тип атаки або нормальна поведінка)
  [ColumnName("PredictedLabel")]
  public string PredictedLabel;

  // Оцінка ймовірностей або довірчий показник для кожного класу
  public float[] Score;
}
```

Додаток Б. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА
на тему:

«Система моніторингу інфраструктури на основі штучного інтелекту»

на здобуття освітнього ступеня магістра
зі спеціальності 122 Комп'ютерні науки
(освітньо-професійної програми Комп'ютерні науки)

Виконав: студент групи КНДМ-63
Горовий Богдан

Керівник: доцент кафедри Комп'ютерних наук
д.т.н. Прокопов С.В.

Київ - 2024

Слайд 1

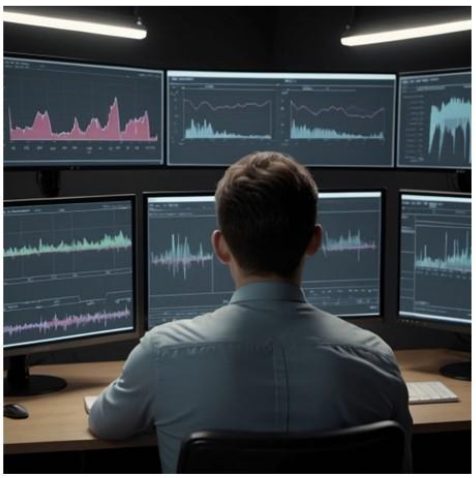
АКТУАЛЬНІСТЬ

Які проблеми постають

- Зростання кількості та складності кібератак
- Часом недостатня ефективність традиційних методів моніторингу
- Людський фактор

Які переваги впровадження ШІ

- Виявлення прихованих закономірностей та аномалій
- Оптимізація витрат
- Своєчасне виявлення загроз



Слайд 2

МЕТА, ЗАВДАННЯ

Наукове завдання – розробка системи моніторингу інфраструктурних об'єктів з використанням штучного інтелекту.

Мета роботи – підвищити рівень безпеки і надійності критичних інфраструктурних систем.

Завдання для досягнення мети

1. Аналіз існуючих систем та принципів їх роботи
2. Вибір та обґрунтування стеку для реалізації системи
3. Розробка системи моніторингу
4. Побудова та навчання моделі машинного навчання
5. Тестування та оцінка

Слайд 3

СИСТЕМИ МОНІТОРИНГУ

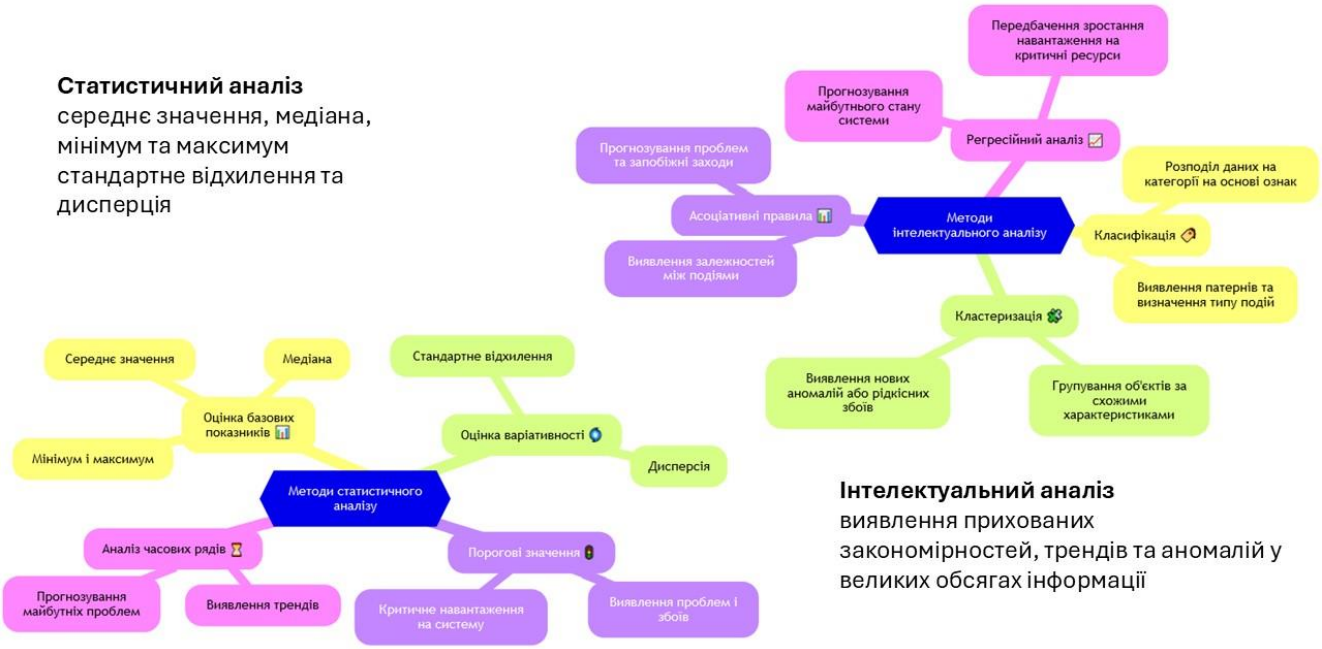
ZABBIX

Nagios® XI™

Слайд 4

ОБРОБКА ТА АНАЛІЗ В МОНІТОРИНГУ

Статистичний аналіз
середнє значення, медіана, мінімум та максимум стандартне відхилення та дисперція

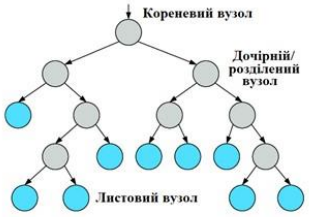


Інтелектуальний аналіз
виявлення прихованих закономірностей, трендів та аномалій у великих обсягах інформації

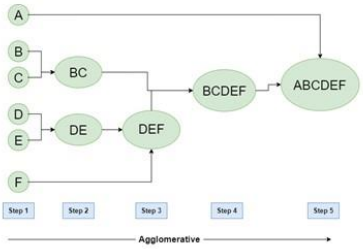
Слайд 5

АЛГОРИТМИ МАШИННОГО НАВЧАННЯ

Методи обробки структурованих даних

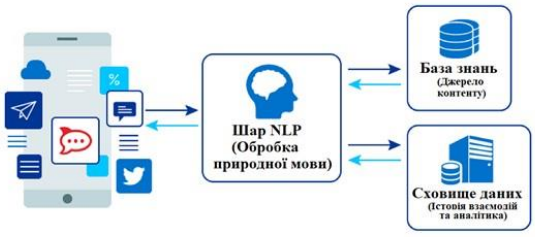


Метод дерева рішень



Метод ієрархічної кластеризації

Методи обробки неструктурованих даних

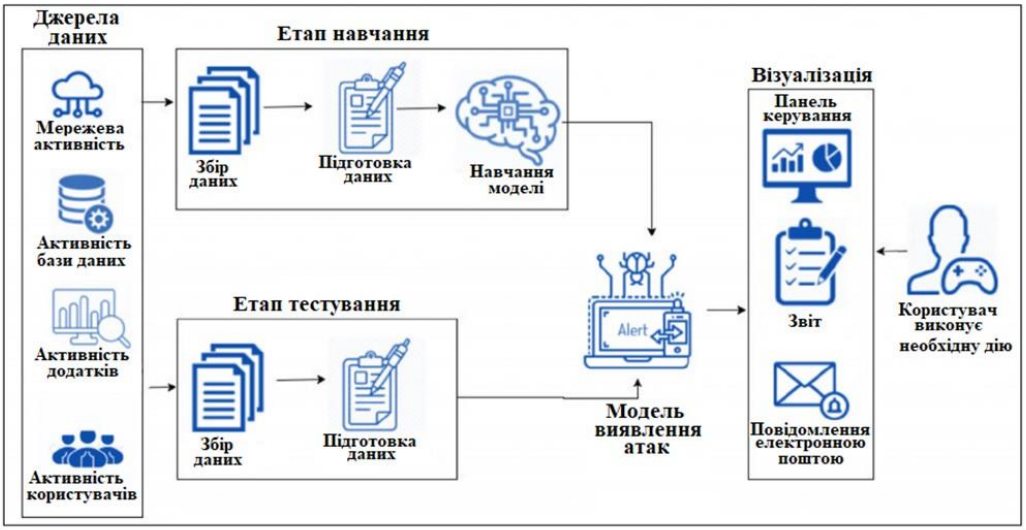


Метод обробки природної мови(NLP)

Метод глибокого навчання

Слайд 6

ШІ ДЛЯ ВИЯВЛЕННЯ АТАК



Загальна схема виявлення атак за допомогою штучного інтелекту

Слайд 7

СТЕК ДЛЯ РОЗРОБКИ: ПОРІВНЯННЯ

Характеристика	Python	Java	C#
Продуктивність (виконання коду)	Середня	Висока	Дуже висока
Підтримка багатопотоковості	Обмежена	Висока	Висока
Переносимість	Висока	Висока	Висока (в межах .NET екосистеми)
Інтеграція з корпоративними системами	Обмежена	Висока	Дуже висока
Простота навчання	Висока	Середня	Середня
Підтримка новітніх технологій	Висока (особливо у сфері AI)	Висока	Дуже висока
Час компіляції	Немає (інтерпретована)	Довгий	Короткий
Спільнота розробників	Дуже велика	Велика	Велика

Характеристика	MS SQL Server	MySQL	FireBird
Підтримка транзакцій	ACID-повна	ACID-повна	ACID-повна
Масштабованість	Висока	Середня	Середня
Розвинені засоби аналітики	Розширені інструменти BI	Обмежені	Обмежені
Інтеграція з іншими системами	Глибока, особливо з продуктами Microsoft	Висока для веб-додатків	Обмежена
Підтримка великих обсягів даних	Висока	Висока	Обмежена
Засоби для забезпечення безпеки	Розширені механізми (ролі, права доступу, шифрування)	Базові можливості безпеки	Базові можливості безпеки
Підтримка багатокористувацького середовища	Розширена	Середня	Обмежена

Слайд 8

СТЕК ДЛЯ НАВЧАННЯ ШІ: ПОРІВНЯННЯ

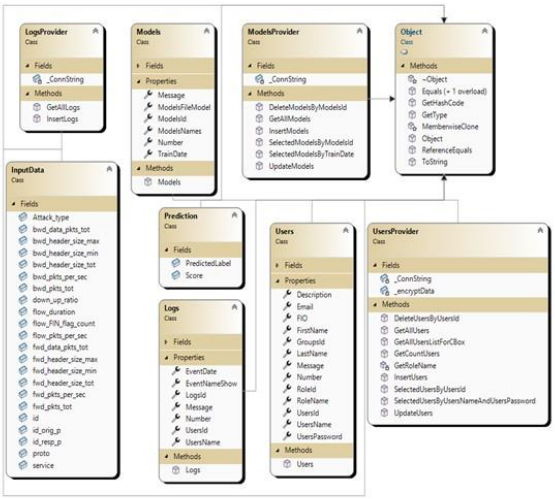
Характеристика	ML.NET	TensorFlow	Keras
Інтеграція з .NET	Повна	Обмежена	Обмежена
Простота використання для C#	Висока	Середня	Середня
Оптимізація для Windows	Висока	Середня	Низька
Локальне розгортання	Швидке та ефективне	Залежить від підтримки обладнання	Залежить від TensorFlow
Підтримка моделей AutoML	Повна	Часткова	Відсутня
Підтримка класичних алгоритмів	Висока	Середня	Низька

Слайд 9

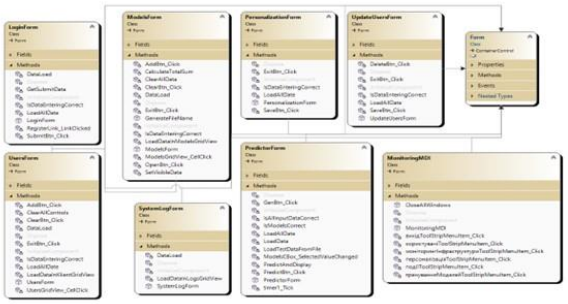
ОБРАНІ ІНСТРУМЕНТИ

Слайд 19

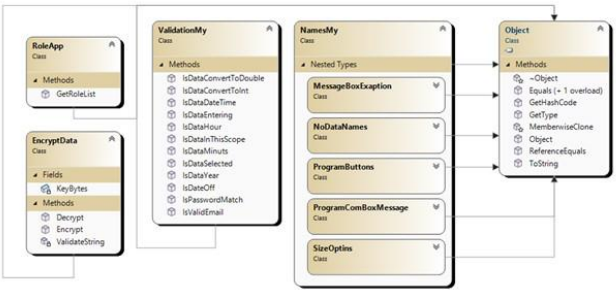
ПРОЕКТУВАННЯ: ТРИРІВНЕВА АРХІТЕКТУРА



Діаграма класів рівня даних



Діаграма класів рівня інтерфейсу системи



Діаграма класів рівня бізнес-логіки

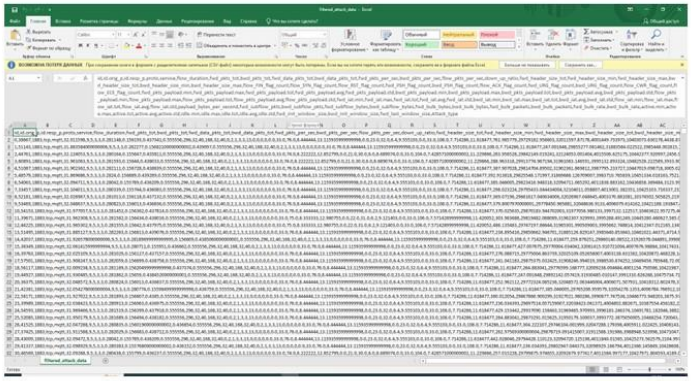
Слайд 11

ДАТАСЕТ ДЛЯ НАВЧАННЯ МОДЕЛІ

Цей датасет сформований на основі промаркованих даних, взятих із авторитетного сайту Kaggle. Містить численні записи з різними параметрами мережевого трафіку, які включають як числові, так і категоріальні ознаки, що характеризують поведінку системи.

Тип атаки	Кількість записів
DOS_SYN_Hping	10,000
Thing_Speak	8,108
ARP_poisioning	7,750
MQTT_Publish	4,146
NMAP_UDP_SCAN	2,590
NMAP_XMAS_TREE_SCAN	2,010
NMAP_OS_DETECTION	2,000
NMAP_TCP_scan	1,002
DDOS_Slowloris	534
Wipro_bulb	253
Metasploit_Brute_Force_SSH	37

Кількість записів по кожному типу атаки

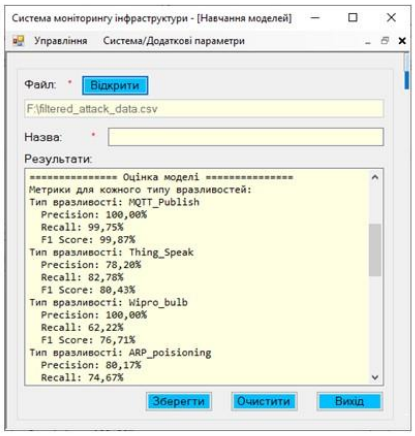


Сформований датасет

Ключова особливість цього набору даних – наявність міток, які вказують на належність кожного запису до конкретного класу, наприклад, нормальної поведінки або аномалії (тип атаки).

Слайд 12

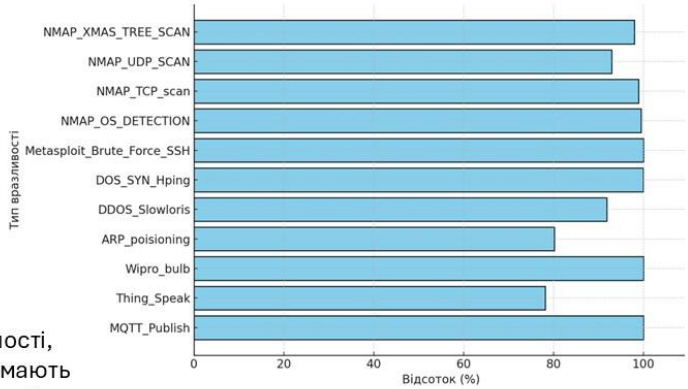
ПРОЦЕС НАВЧАННЯ МОДЕЛІ ТА ОЦІНКА



Процес навчання моделі

Для оцінки якості моделі використовувались метрики:

- **Точність (Accuracy):** відсоток правильно класифікованих зразків.
- **Повнота (Recall):** відсоток правильно виявлених атак серед усіх фактичних атак.
- **F1-оцінка (F1-score):** гармонійне середнє між точністю та повнотою.

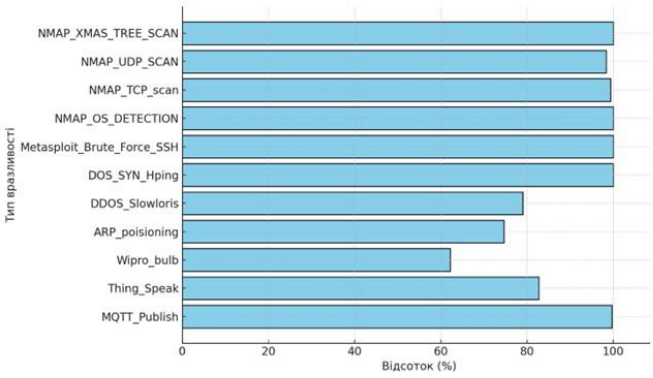


Точність для кожного типу вразливості

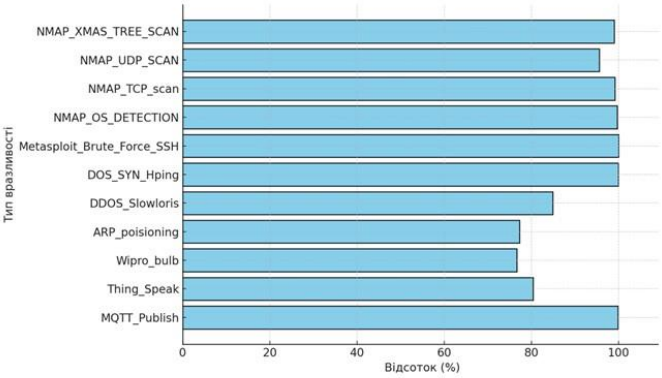
Для більшості атак модель досягає високої точності, наближеної до 100%. Водночас деякі типи атак мають нижчі показники, що може бути наслідком меншої кількості записів у датасеті для цих класів

Слайд 13

ОЦІНКА МОДЕЛІ



Повнота для кожного типу вразливості



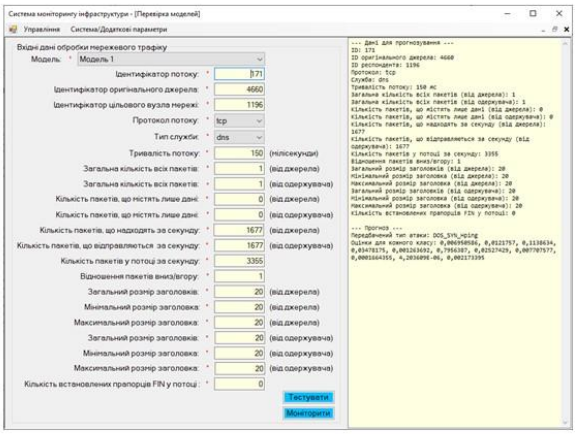
F1-оцінка для кожного типу вразливості

Загалом, результати підтверджують, що модель добре справляється із завданням розпізнавання поширених типів атак, однак для менш представлених класів необхідно проводити додаткове доопрацювання, зокрема збільшення вибірки або адаптацію параметрів моделі

Слайд 14

СЦЕНАРІЙ ВИЯВЛЕННЯ АНОМАЛІЙ

Система отримала мережевий потік із наступними характеристиками: протокол TCP, служба DNS, тривалість потоку 150 мс, кількість пакетів від джерела та одержувача по одному, а швидкість передачі даних для обох напрямків становила 1677 пакетів/сек. Загальний розмір заголовків у потоці – 20 байтів.
Відсутність прапорців FIN у потоці свідчить про нехарактерний завершення з'єднання, що могло слугувати однією з ознак атаки



На основі аналізу цих даних модель передбачила тип атаки DOS_SYN_Hping, з ймовірністю 79.56%. Висока ймовірність вказує на те, що модель впевнено ідентифікувала цей потік як частину атаки типу DoS, характерної для SYN-флуду

Слайд 15

ВИСНОВКИ

1. Проаналізовано існуючі системи моніторингу та методи машинного навчання.
2. Обґрунтовано вибір технологічного стеку (C#, ML.NET, MS SQL Server).
3. Реалізовано систему моніторингу з трирівневою архітектурою.
4. Навчено модель на збалансованому датасеті, досягнувши високих показників точності, повноти та F1-оцінки.
5. Тестування підтвердило ефективність системи у виявленні атак.

Слайд 16