Міністерство освіти і науки України

Національний технічний університет
«Дніпровська політехніка»

Навчально-науковий
інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня бакалавра

здобувача Давиденко Матвій Максимович
(ПІБ)

академічної групи 123-21з-1
(шифр)

спеціальності 123 Комп'ютерна інженерія
(код і назва спеціальності)

за освітньо-професійною програмою Комп'ютерна інженерія
(офіційна назва)

на тему "Нейромережевий аналіз структури інтернет-трафіку для класифікації мережевої активності"
(назва за наказом ректора)

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
кваліфікаційної роботи	доц. Селівьорстова Т.В.			
загального розділу	доц. Селівьорстова Т.В.			
спеціального розділу	доц. Селівьорстова Т.В.			
Рецензент				
Нормоконтролер	проф. Цвіркун Л.І.			

Дніпро
2025

ЗАТВЕРДЖЕНО:
завідувач кафедри
інформаційних технологій
та комп'ютерної інженерії
(повна назва)
Гнатушенко В.В.
(підпис) (прізвище, ініціали)

"25" січня 2025 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня бакалавр

здобувача Давиденко М. М. **академічної групи** 123-21з-1
(прізвище та ініціали) (шифр)

спеціальності 123 Комп'ютерна інженерія

за освітньо-професійною програмою Комп'ютерна інженерія
(офіційна назва)

на тему "Нейромережевий аналіз структури інтернет-трафіку для класифікації мережевої активності"

затверджену наказом ректора НТУ «Дніпровська політехніка» від 05.05.2025 № 337-с

Розділ	Зміст	Термін виконання
Загальний розділ	На основі матеріалів виробничих практик, інших науково-технічних джерел показати актуальність завдання, сформулювати мету та задачі виконання кваліфікаційної роботи	10.02.2025
Спеціальний розділ	Сформулювати найменування й призначення веб-орієнтованної системи, висунути технічні вимоги до нього	15.03.2025
	Розробити архітектуру веб-орієнтованної системи, обґрунтування технічних характеристик.	20.04.2025
	Реалізувати програмний модуль, використовуючи відповідні інструменти програмування та враховуючи принципи розробки надійних та ефективних програмних систем, що є частиною комп'ютерної інженерії	07.05.2025
	Протестувати розроблену веб-орієнтовану систему на предмет її функціональності, продуктивності обміну даними в реальному часі та стійкості до можливих збоїв	31.05.2025

Завдання видано _____ доц. Селівьорстова Т.В.
(підпис керівника) (прізвище, ініціали)

Дата видачі 25.01.2025

Дата подання до екзаменаційної комісії _____

Прийнято до виконання _____ Давиденко М. М.

РЕФЕРАТ

Пояснювальна записка: 64 с., 27 рис., 2 табл., 1 додаток, 15 джерел.

Об'єкт роботи – процеси виявлення мережевих атак у комп'ютерних мережах на основі методів машинного навчання..

Предмет роботи – методи, алгоритми та програмні засоби для аналізу інтернет-трафіку з метою виявлення мережевих атак у режимах реального часу та офлайн.

Метою роботи є розробка та реалізація програмного застосунку для автоматизованого аналізу мережевого трафіку в режимі реального часу та офлайн, використовуючи методи машинного навчання, для ефективного виявлення аномалій та потенційних кібератак.

Завдання:

1 Проаналізувати предметну область, існуючі методи та інструменти для аналізу мережевого трафіку та виявлення атак.

2 Підготувати та попередньо обробити набір даних мережевого трафіку, придатний для навчання моделей машинного навчання.

3 Вибрати та обґрунтувати застосування відповідних моделей машинного навчання для класифікації мережевого трафіку (наприклад, PHM, LSTM, ансамблеві методи: Random Forest, CatBoost, SVM).

4 Реалізувати обрані моделі машинного навчання та адаптувати їх для задачі виявлення атак.

5 Спроектувати архітектуру системи аналізу трафіку, що забезпечує роботу в режимах реального часу та офлайн.

6 Реалізувати програмну частину спроектованої системи, включаючи аналізатор пакетів та користувацький інтерфейс.

7 Виконати тестування розробленого застосунку та оцінити якість роботи моделей машинного навчання (функціональне тестування, A/B тестування).

Ключові слова: рафік, машинне навчання, виявлення атак, кібербезпека, рекурентні нейронні мережі, офлайн-аналіз, PCAP.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАК, ОДИНИЦЬ ТА ТЕРМІНІВ

API – Application Programming Interface (Інтерфейс прикладного програмування)

BPF – Berkeley Packet Filter (Фільтр пакетів Берклі)

CPU – Central Processing Unit (Центральний процесорний пристрій)

DDoS – Distributed Denial of Service (Розподілена відмова в обслуговуванні)

DoS – Denial of Service (Відмова в обслуговуванні)

GPU – Graphics Processing Unit (Графічний процесорний пристрій)

HTTP – HyperText Transfer Protocol (Протокол передачі гіпертексту)

IDS – Intrusion Detection System (Система виявлення вторгнень)

IP – Internet Protocol (Інтернет-протокол)

LSTM – Long Short-Term Memory (Довга короткочасна пам'ять)

ML – Machine Learning (Машинне навчання)

OS – Operating System (Операційна система)

PCAP – Packet Capture (Захоплення пакетів)

РНМ – Рекурентна нейронна мережа (RNN – Recurrent Neural Network)

RPCAPD – Remote Packet Capture Daemon (Демон віддаленого захоплення пакетів)

SSH – Secure Shell (Безпечна оболонка)

SVM – Support Vector Machine (Метод опорних векторів)

TDD – Test-Driven Development (Розробка через тестування)

TCP – Transmission Control Protocol (Протокол керування передачею)

UI – User Interface (Користувацький інтерфейс)

UDP – User Datagram Protocol (Протокол дейтаграм користувача)

URL – Uniform Resource Locator (Уніфікований покажчик ресурсу)

Wi-Fi – Wireless Fidelity

ВСТУП

Зі стрімким розвитком інформаційних технологій та глобалізацією цифрового простору, інтернет-трафік перетворився на життєво важливу складову сучасного суспільства. Він є основою для функціонування незліченної кількості сервісів, додатків та комунікацій, від яких залежить повсякденна діяльність мільярдів користувачів і стабільна робота критичної інфраструктури. Зростання обсягів трафіку, його постійна еволюція та зростаюча складність вимагають застосування все більш досконалих методів для його аналізу та класифікації. Це необхідно не лише для оптимізації мережевих ресурсів, підвищення якості обслуговування (QoS) та виявлення аномалій, але й для ефективної протидії кіберзагрозам, забезпечення мережевої безпеки та реалізації механізмів контролю доступу.

Традиційні підходи до аналізу мережевої активності, що базуються на статичних правилах та сигнатурному аналізі, часто виявляються неефективними в умовах динамічно мінливого та зашифрованого трафіку. Вони нездатні своєчасно реагувати на нові види атак, адаптуватися до змін у поведінці користувачів та додатків, а також ідентифікувати приховану активність. У зв'язку з цим, виникає гостра потреба у розробці та впровадженні інноваційних методик, які б дозволяли здійснювати глибокий, динамічний та адаптивний аналіз структури інтернет-трафіку.

Одним з найбільш перспективних напрямків у цій галузі є застосування нейронних мереж. Завдяки своїй здатності до самонавчання, виявлення складних прихованих закономірностей у великих обсягах даних та адаптації до нових умов, нейромережі відкривають широкі можливості для революціонізування процесів класифікації мережевої активності. Вони дозволяють автоматизувати виявлення аномалій, розрізняти легітимний трафік від шкідливого, ідентифікувати специфічні типи додатків та поведінкових патернів, що є критично важливим для побудови гнучких, надійних та безпечних мережевих систем.

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Актуальність

Аналіз мережевого трафіку є критично важливою задачею в багатьох сферах діяльності. Цей аналіз може мати різні цілі: від виділення основних пристроїв, з яких користувачі отримують доступ до сайту, та розуміння джерел трафіку, до виявлення можливих вторгнень або мережевих атак. Остання задача є особливо гострою в сучасних умовах. З кожним роком людство все активніше використовує інтернет у своєму житті, зростає кількість «розумних» пристроїв, активно розвиваються хмарні технології. Тому забезпечення безпеки та надійності сервісів стає критичним завданням для багатьох компаній.

Згідно зі звітами компаній Kaspersky та Securelist, кількість та якість DDoS-атак щороку зростає. У період з першого кварталу 2020 року по третій квартал 2022 року кількість проведених атак збільшилася майже у п'ять разів [1–3]. Динаміка зростання кількості атак за цей період представлена на рисунку 1.1.

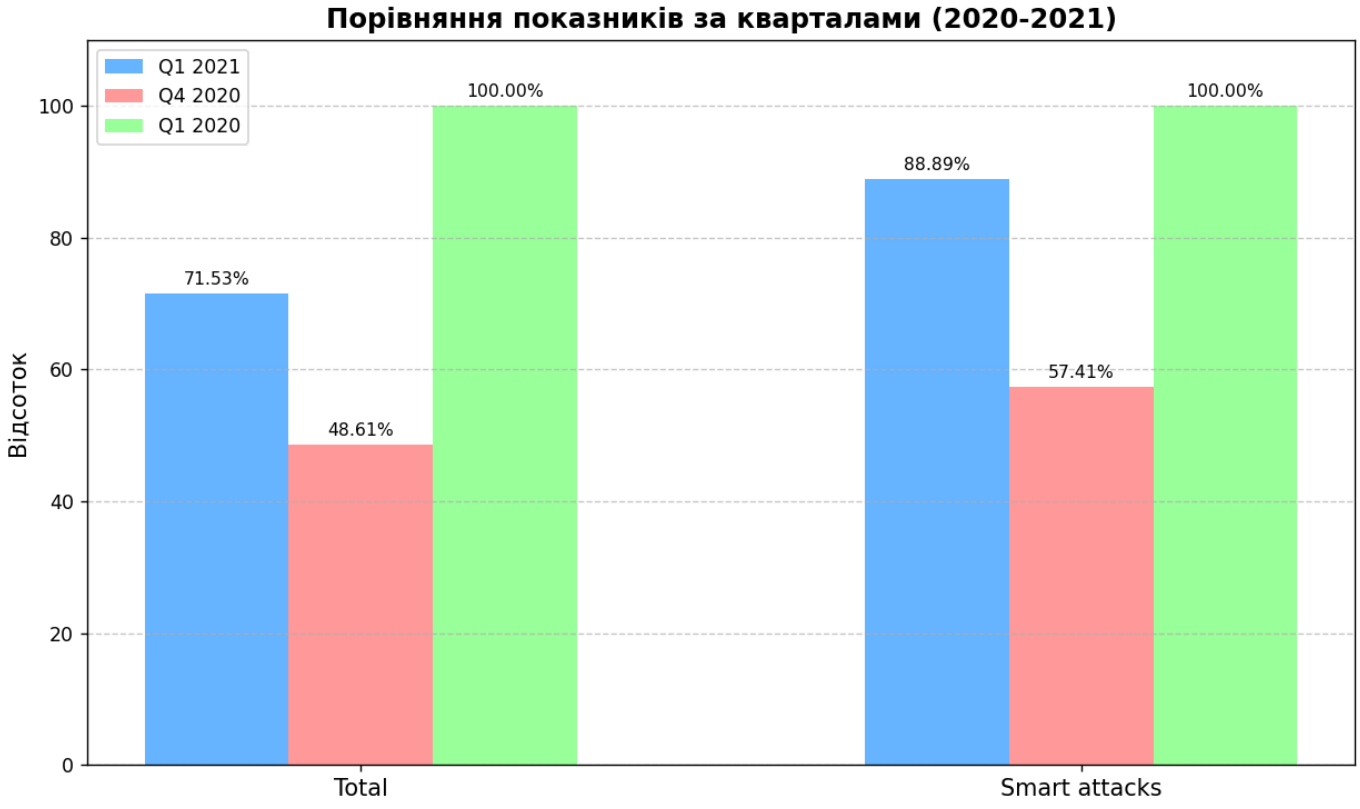

Рисунок 1.1 – Порівняння показників кількості атак за кварталами

У 2023 році тенденція зростання кількості DDoS-атак продовжилася. Згідно зі звітами компанії DDoS-Guard, загальна кількість атак у 2023 році зросла майже в 2 рази. Компанія також відзначає значне зростання кількості пристроїв, з яких здійснюються атаки. Серед таких пристроїв компанія виділяє смарт-телевізори, точки доступу, камери відеоспостереження. Тенденція використання таких пристроїв зберігається останні кілька років. Розподіл атак за місяцями року та порівняння з минулим роком представлено на рисунку 1.2 [4].

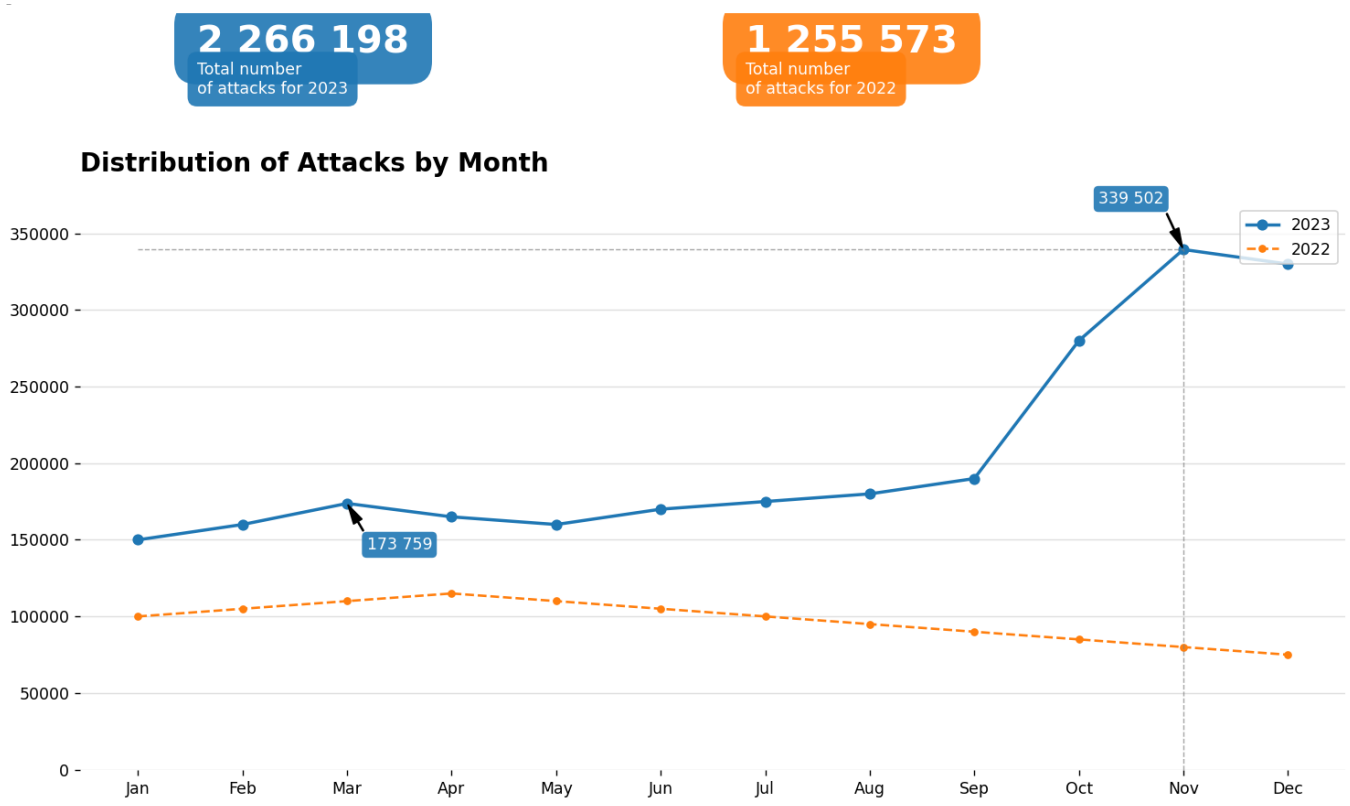

Рисунок 1.2 – Порівняння кількості атак за 2022 та 2023 роки

Згідно зі звітом компанії CLOUDFLARE, у 2023 році також спостерігалося значне зростання потужності проведених атак. Під час найбільшої DDoS-атаки, зафіксованої компанією у 2023 році, на сервіс надходив 201 мільйон запитів на секунду, що майже у 8 разів більше, ніж під час найбільшої атаки у 2022 році [3]. Порівняльний звіт представлено на рисунку 1.3.

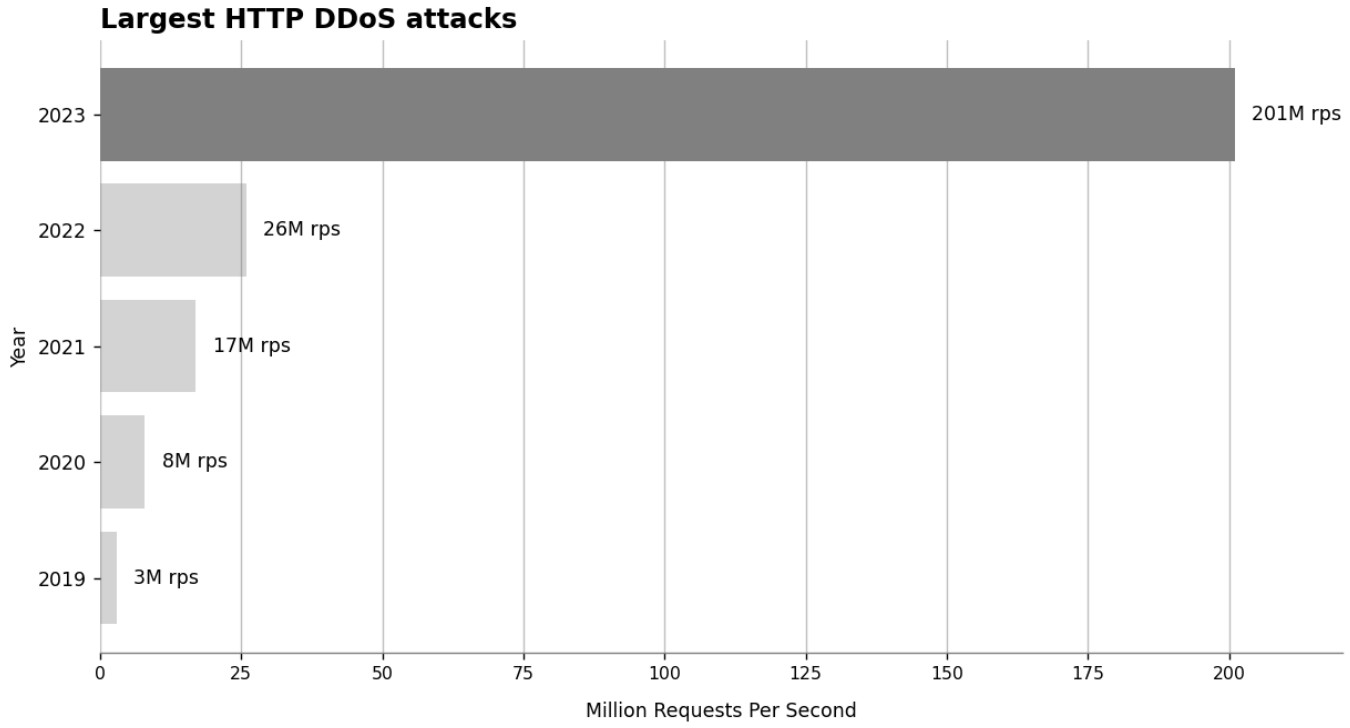

Рисунок 1.3 – Порівняння по кількості запитів на секунду

Метою даної кваліфікаційної роботи є створення програмного забезпечення для аналізу мережевого трафіку в реальному часі із застосуванням методів машинного навчання.

Для досягнення цієї мети необхідно вирішити такі завдання:

1. Здійснити аналіз предметної області, що стосується мережевого трафіку та методів його обробки.
2. Зібрати та підготувати відповідний набір даних, необхідний для навчання та тестування моделей.
3. Вибрати оптимальні моделі машинного навчання, що відповідають вимогам до аналізу мережевої активності.
4. Здійснити програмну реалізацію обраних моделей машинного навчання.
5. Розробити архітектуру системи для аналізу трафіку в реальному часі.
6. Реалізувати розроблену систему для аналізу інтернет-трафіку в реальному часі.
7. Провести всебічне тестування системи аналізу інтернет-трафіку в

реальному часі для оцінки її ефективності та надійності.

1.2 Архітектура NetGPT

У дослідженні [2] автори застосували підхід, що базується на великих попередньо навчених моделях, для вирішення низки завдань, пов'язаних з аналізом мережевого трафіку. Цей підхід мав на меті подолати проблему використання багатьох невеликих моделей, кожна з яких була б навчена для окремого завдання на специфічних наборах даних, що могли бути недостатньо великими для успішного навчання моделей.

Для вирішення поставленого завдання авторами було запропоновано архітектуру під назвою NetGPT. Архітектура NetGPT складається з двох ключових частин: попередньо навченої частини та частини для доналаштування.

У процесі попереднього навчання мережевий трафік кодується у прихований простір ознак. Для забезпечення узагальнювальної здатності моделі, у попередньо навченій частині використовувалися нерозмічені записи. У процесі доналаштування збільшується обсяг використовуваної інформації шляхом перемішування заголовків, сегментації семантики мережевих пакетів під час навчання, включення необхідних для вирішення різних завдань міток, а також генерації трафіку на основі отриманих даних.

Запропонована архітектура NetGPT вирішує проблеми неоднорідності реального трафіку та різноманітності можливих завдань завдяки використанню попередньо навченої частини та доналаштування.

Як базова модель для процесу попереднього навчання використовується GPT-2. Для навчання генеративним завданням також застосовуються моделі з сімейства GPT.

Архітектура попередньо навченої частини представлена на рисунку 1.4.

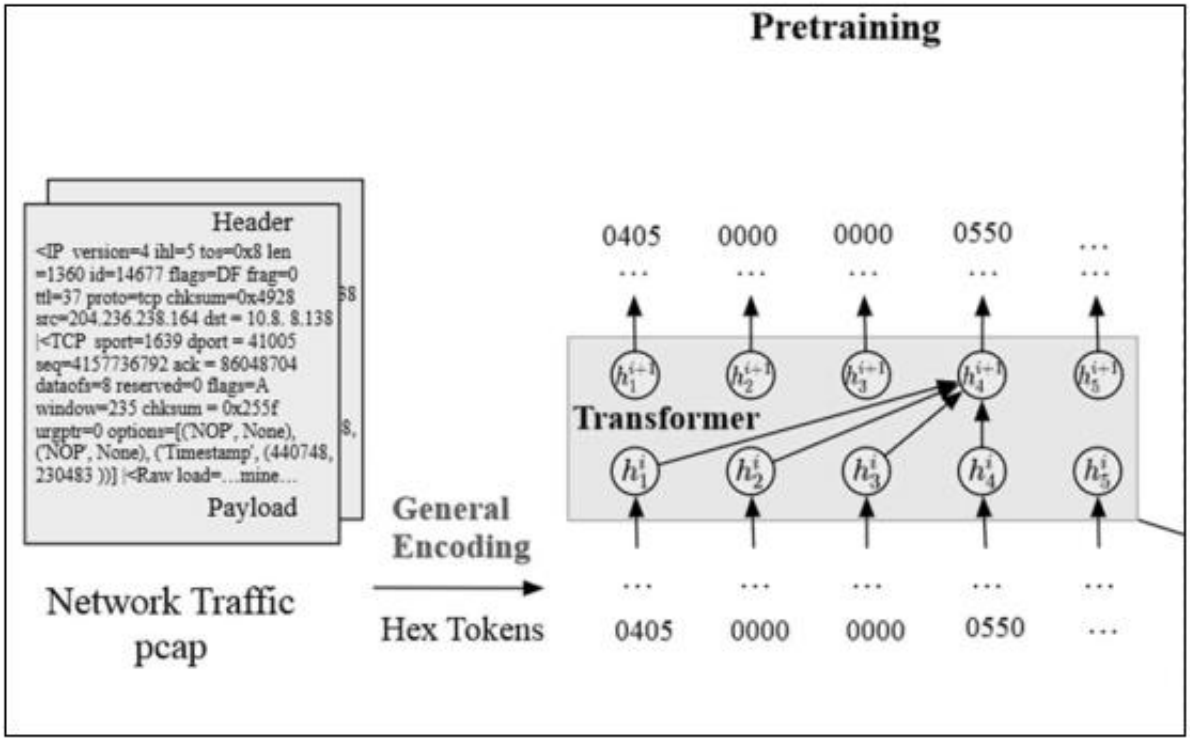

Рисунок 1.4 – Архітектура попередньо навченої моделі

Архітектура доналаштовувальної частини мережі представлена на рисунку 7. Для навчання та тестування моделі було використано п'ять наборів даних: ISXW, DoHBrw, USTCTFC, PrivII, Cybermine. Розподіл кількості записів цих наборів даних представлено на рисунку 1.5. Набір даних Cybermine не використовувався для навчання, а був представлений лише в тестуванні. Було сформульовано вісім завдань, яким модель мала навчитися, серед них: виявлення VPN та класифікація застосунків на наборі даних ISXW; виявлення трафіку DNS-over-HTTPS та генерація запитів DoH для набору даних DoHBrw; виявлення мережевих атак та ідентифікація застосунків у наборі даних USTCTFC; виявлення кібермайнінгу та ідентифікація криптовалют для набору Cybermine. Для тестування використовувалися метрики точності та макро F1-міри, а також дивергенція Єнсена-Шеннона (JSD). Валідація та тестування проводилися на випадкових вибірках пакетів розміром 5 000. Для задач класифікації трафіку модель навчалася протягом 50 епох, а для задач генерації трафіку – протягом 10 епох.

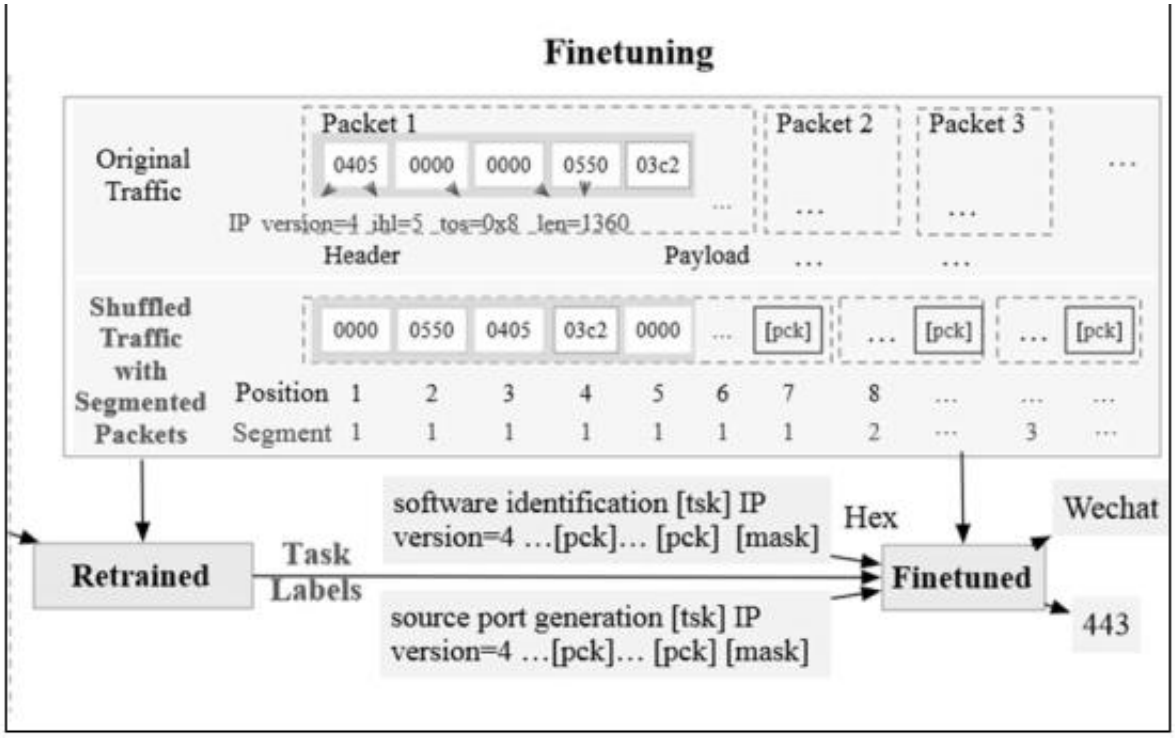

Рисунок 1.5 – Архітектура донавченої моделі

Середня точність архітектури NetGPT на задачі аналізу пакетів склала 98,56%, а середня точність на задачі аналізу мережевих потоків – 94,6%. Для порівняння, архітектура ET-Bert на тих самих задачах досягає точності 98,25% та 90,8% відповідно.

Dataset	ISXW	DoHBrw	USTCTFC	PrivII	Cybermine
# packets	1,338,300	77,149,018	1,010,534	38,250,867	7,862
# flows	1,262	2,497	5,039	36,926	2,617

Рисунок 1.5 – Набір даних

Для аналізу ефективності моделей у задачах генерації трафіку використовувалася метрика JSD. Метрика JSD здатна відображати достовірність отриманих даних у задачах генерації, порівнюючи реальні дані та дані, отримані в процесі генерації, для розподілу, що відповідає предметній області. Середнє значення метрики JSD для моделі NetGPT склало 0,0406, тоді як для моделі GPT-2 – 0,0417. Виходячи з отриманих результатів, автори роблять висновок про високу ефективність

реалізованої моделі у задачах генерації трафіку. NetGPT демонструє кращу середню продуктивність та перевершує GPT-2 на більшості наборів даних, що свідчить про ефективність застосування техніки перемішування при навчанні генеративним завданням. Високі результати на наборі даних Cybermine підтверджують високу узагальнювальну здатність моделі та її спроможність вирішувати супутні задачі.

Автори також роблять висновок про високий ступінь відповідності згенерованого трафіку вихідному набору даних. Розподіл полів заголовків трафіку, отриманого за допомогою NetGPT, збігається з реальними даними.

1.3 Система OMINACS

У науковій праці [3] автори зосереджуються на вирішенні проблеми швидкого застарівання навчених моделей, які застосовуються в системах виявлення загроз у мережах. Для подолання цього виклику був запропонований інноваційний метод навчання в реальному часі, що інтегрує декілька моделей машинного навчання. Загальна схема функціонування цієї системи наведена на рисунку 1.6.

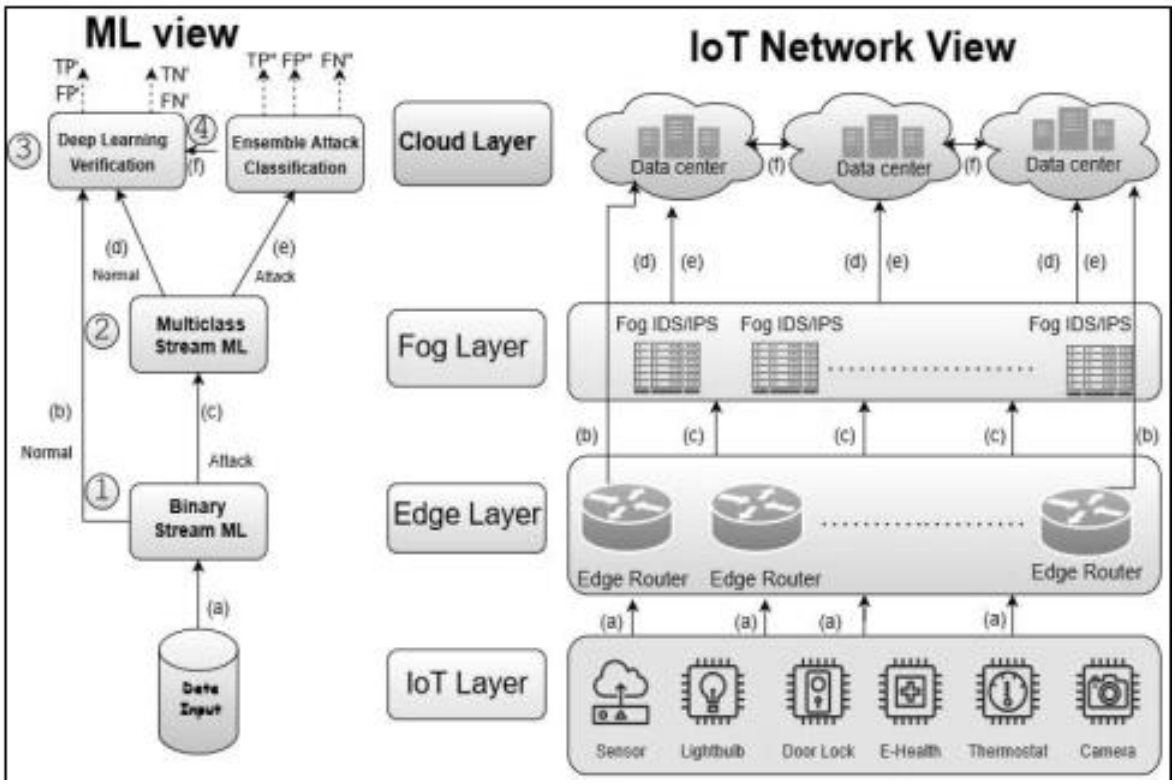

Рисунок 1.6 – Схема системи OMINACS

Система OMINACS складається з чотирьох основних компонентів. Перший компонент реалізує алгоритм потокового машинного навчання, який базується на адаптивних деревах Хоффдінга. Цей компонент використовується для бінарної класифікації, тобто для визначення, чи є трафік звичайним, чи він містить загрозу. Залежно від результату цієї класифікації, відповідні записи передаються для подальшої обробки до другого або третього компонента системи.

Другий компонент також реалізує алгоритм потокового машинного навчання, але його призначення полягає в класифікації конкретного типу атаки, якщо трафік було визначено як шкідливий.

Третій компонент відповідає за перевірку даних, які були класифіковані як звичайний трафік, використовуючи для цього модель глибокого навчання. На вхід третього компонента надходять дані, які позначені як звичайний трафік, з першого, другого та четвертого компонентів системи.

У четвертому компоненті здійснюється перевірка відповідності класів атак за допомогою використання множини моделей. Кожна з цих моделей визначає достовірність присвоєного класу, перевіряючи, чи є присвоєний клас коректним, чи це помилка класифікації. На цьому етапі для класифікації застосовуються ансамблеві моделі.

Для оцінки якості навчених моделей були використані такі метрики: точність (Accuracy), прецизія (Precision), показник істинно позитивних спрацьовувань (True Positive Rate), показник хибних спрацьовувань (False Alarm Rate) та F1-міра. Для навчання та тестування системи були застосовані набори даних BoT-IOT, TON-IOT та CIC-IOT-2022. Результати навчання та тестування представлені на рисунку 1.7.

Dataset	ACC%	Prec%	TPR%	FAR%	F1%
BOT-IOT	98.91	96.76	99.88	03.24	98.46
TON-IOT	94.33	90.45	99.83	9.55	94.35
IOT-22	96.55	96.97	99.89	03.03	97.16

Рисунок 1.7 – Результати тестування

Згідно з рисунком 1.7, результати навчання та тестування показують, що система досягає точності понад 90% при частоті хибних спрацьовувань менше 3% на всіх представлених наборах даних. Найвищий результат за метрикою точності (Accuracy) був досягнутий на наборі даних BOT-IOT і склав 98,91%. Це свідчить про високу ефективність розробленої системи.З огляду на експоненційне зростання локальних мереж, зростання кількості кібератак та необхідність моніторингу мережевого трафіку для забезпечення безпеки, розробка ефективної методики віддаленого перехоплення є вкрай важливою. Існуючі дослідження мають значні прогалини у цьому аспекті, зокрема, відсутність повноцінних описів методик та оцінки їхньої ефективності.

1.4 Порівняльний аналіз методів машинного навчання для класифікації мережевого трафіку

У роботі [5] автори здійснюють порівняльний аналіз ефективності чотирьох алгоритмів машинного навчання при розв'язанні задачі класифікації інтернет-трафіку. Для проведення дослідження вони використовують набір даних NIMS. Цей набір даних містить інформацію про шість різних типів мережевих застосунків: LFWD, RFWD, SCP, SFTP, SHELL та X11. Важливо зазначити, що для кожного типу застосунку була отримана однакова кількість записів, а загальна кількість записів у всьому наборі даних NIMS становила 14 681.

Для свого аналізу автори обрали чотири алгоритми машинного навчання: метод опорних векторів (SVM), наївний баєсівський класифікатор, метод k-найближчих сусідів (K-NN) та дерево рішень C4.5. Для відбору значущих ознак застосовувалися такі методи, як Information Gain (Приріст інформації), ReliefF, Symmetrical Uncertainty (Симетрична невизначеність) та Gain Ratio (Відношення приросту). Якість навчених моделей оцінювалася за допомогою матриць сплутаності, а також метрик Accuracy (Точність), Precision (Прецизія), Recall (Повнота) та F1-міри.

Результати, отримані без попереднього відбору ознак, представлені на рисунку 13. У цьому випадку найвищу точність показав алгоритм дерев рішень C4.5,

досягнувши показника точності 98,25% при найменшому часі виконання.

В ході процесу відбору ознак з набору даних NIMS було виділено 15 найбільш значущих атрибутів. Ці відібрані ознаки та методи, за допомогою яких вони були отримані, відображені на рисунку 1.8.

FS Method	Feature Names	Feature Values/Rankings
CFS	{maxbpktl, stdbpktl, stdfpktl, maxfpktl, stdfiat, meanfpktl, stdbiat, meanfiat, meanbkptl, meanbiat}	{0.383, 0.363, 0.327, 0.326, 0.325, 0.318, 0.310, 0.305, 0.298, 0.269}
GR	{maxfpktl, maxbpktl, minfiat, meanbkptl, meanfpktl, stdbpktl,maxfiat, maxbiat, stdfpktl, totalfvol}	{0.937,0.588,0.483,0.453,0.442, 0.403,0.353,0.348,0.334,0.293}
IG	{maxbiat, maxfiat, meanfpktl, stdbpktl, meanbkptl, stdfpktl, minfiat, meanfiat,minbiat, stdbiat}	{2.476, 2.475,1.992, 1.960, 1.724, 1.610, 1.572, 1.428, 1.425, 1.410}
RELIEFF	{meanfpktl, stdbpktl, maxfpktl, stdfpktl, maxfiat, maxbpktl, durattion, maxbiat, stdbiat, stdfiat, meanbkptl}	{0.203, 0.181, 0.174, 0.169, 0.153, 0.136, 0.127, 0.122, 0.112, 0.111}
SU	{maxfpktl, meanfpktl, meanbkptl, minfiat, stdbpktl, maxfiat, maxbiat, maxbpktl, stdfpktl, stdbiat}	{0.617, 0.562, 0.54, 0.539, 0.526, 0.515, 0.510, 0.455, 0.435, 0.370}

Рисунок 1.8 – Вибір ознак

Після відбору значущих ознак класифікатори були повторно навчені. Результати навчання до та після відбору ознак представлені на рисунку 1.9. Застосування відбору значущих ознак призвело до зростання як точності, так і швидкості роботи всіх використаних алгоритмів. Найкращим алгоритмом, як і раніше, залишилися дерева рішень C4.5, які досягли вражаючої точності 99,81% та швидкості виконання 0,07 секунд.

Classifier	Accuracy	Precision	Recall	F-Measure	Runtime(s)
NB	85.64	87.3	87.1	85.8	0.38
NB-FS	87.34	88.8	89.3	89.4	0.16
KNN	98.12	99.3	98.3	96	0.18
KNN-FS	99.72	99.7	99.7	99.7	0.09
SVM	94.47	93.4	95.1	95.2	1.35
SVM-FS	95.58	92.3	92.7	91.2	0.65
C45	98.25	97.5	98.4	98.3	0.16
C4.5-FS	99.81	99.8	99.8	99.8	0.07

Рисунок 1.9 – Результати навчання моделей до і після відбору ознак

1.5 Виявлення атак відмови в обслуговуванні за допомогою алгоритмів машинного навчання

У статті [6] розглядаються два основні підходи до ідентифікації DDoS-атак. Перший підхід полягає у розробці математичної моделі, яка визначає взаємозв'язок між часом надходження запитів та пропускною здатністю, з подальшим аналізом отриманих даних. Як моделі машинного навчання у цьому підході були використані логістична регресія та наївний баєсівський класифікатор.

Запропонована математична модель для класифікації атак використовує дві ключові характеристики: швидкість з'єднання та пропускну здатність. Швидкість з'єднання – це характеристика, що відображає потенційний обсяг даних, який може бути переданий каналом зв'язку за одиницю часу. Пропускна здатність, у свою чергу, показує фактичну кількість даних, які були успішно передані від джерела до призначення.

Для тестування моделі використовувався набір даних CAIDA 2007. Кожні 10 секунд на вхід моделі подавалися дані з 20 090 різних IP-адрес.

Для навчання моделей машинного навчання також застосовувався набір даних CAIDA 2007. Цей набір даних був розділений таким чином: 80% для навчання, 20% для тестування та 10% для валідації.

Для оцінки якості запропонованої математичної моделі використовувалася метрика Miss Rate, яка розраховується за формулою (1.1):

$$\text{Miss Rate} = \frac{\text{False Negative}}{\text{True Positive} + \text{False Negative}} \quad (1.1)$$

де False Negative – кількість записів, помилково класифікованих як негативні (тобто атака не була виявлена, коли вона була); True Positive – кількість записів, коректно класифікованих як позитивні (тобто атака була правильно виявлена).

Значення метрики Miss Rate на тестовому наборі даних склало 0,025. Це означає, що лише 2,5% реальних атак були пропущені моделлю.

Для оцінки якості навчання моделей машинного навчання додатково використовувалися метрики Accuracy (точність), Mean Absolute Error (MAE – середня абсолютна похибка) та Recall (повнота). В результаті навчання точність моделі логістичної регресії склала 99,83%, тоді як точність класифікатора наївного Байєса – 98,67%. Значення метрики MAE для класифікатора наївного Байєса становило 0,0163, а для моделі логістичної регресії – 0,0017. Значення метрики Recall склало 0,998 для моделі логістичної регресії та 0,99 для наївного баєсівського класифікатора. На основі цих результатів був зроблений висновок, що модель логістичної регресії демонструє кращу продуктивність у задачі класифікації мережевого трафіку.

1.6 Задача класифікації трафіку

У сфері машинного навчання завдання класифікації визначається як процес побудови алгоритму. Цей алгоритм, використовуючи заздалегідь визначений набір об'єктів, що вже розподілені за класами на основі їхніх ознак, має бути здатним визначати клас довільного об'єкта як з початкового набору даних, так і для нових, раніше невідомих об'єктів [1].

У контексті мережевого трафіку, завдання класифікації може бути сформульоване як: визначення типу трафіку на основі його вхідних характеристик. Такими вхідними характеристиками можуть виступати як безпосередньо дані пакетів, так і різноманітні частотні показники. Результатом такої класифікації може бути ідентифікатор конкретної програми, яка генерує трафік, або ж ідентифікатор загального виду трафіку (наприклад, веб-трафік, відеопотік тощо) [3].

Для забезпечення передачі даних між різними мережевими пристроями використовується мережева модель TCP/IP, яка складається з чотирьох основних рівнів:

– канальний рівень описує механізми обміну інформацією на рівні мережевих пристроїв (наприклад, комутаторів) та визначає дані, що передаються від одного пристрою до іншого. Цей рівень також відповідає за розрахунок максимальної відстані передачі пакетів, частоти сигналу та затримки відповіді. Головним протоколом канального рівня є Ethernet [3];

– мережевий рівень відповідає за надсилання мережевих пакетів та керування їхнім переміщенням по мережі для забезпечення успішної доставки даних до отримувача [12];

– транспортний рівень забезпечує надійне з'єднання між джерелом та отримувачем даних. На цьому рівні дані поділяються на пакети та нумеруються для формування послідовностей. Транспортний рівень визначає обсяг даних для надсилання, пункт призначення та швидкість з'єднання. Він гарантує безпомилкове надсилання даних у правильній послідовності та сповіщає про отримання відправлених пакетів даних отримувачем;

– прикладний рівень стосується програмного забезпечення, що потребує стеку технологій TCP/IP для взаємодії. Це рівень, на якому користувачі найчастіше працюють з такими системами, як електронна пошта або платформи обміну повідомленнями. Він містить безліч протоколів, зокрема HTTP, HTTPS, SSH, NTP, FTP, SMTP та інші [7].

Згідно з класифікацією CISCO, існують два основні методи класифікації мережевого трафіку: метод, що базується на аналізі вмісту інтернет-пакетів, та метод, що ґрунтується на статистичному аналізі поведінки трафіку. Ця класифікація візуально представлена на рисунку 1.10 [8].

Найбільш поширеним методом класифікації є аналіз вмісту інтернет-пакетів. Цей метод поділяється на базовий та розширений аналіз вмісту.

Базовий підхід до класифікації ґрунтується на аналізі інформації, що міститься в IP-заголовках. У якості даних можуть використовуватися IP-адреса, MAC-адреса, а

також протоколи, що застосовуються для передачі.

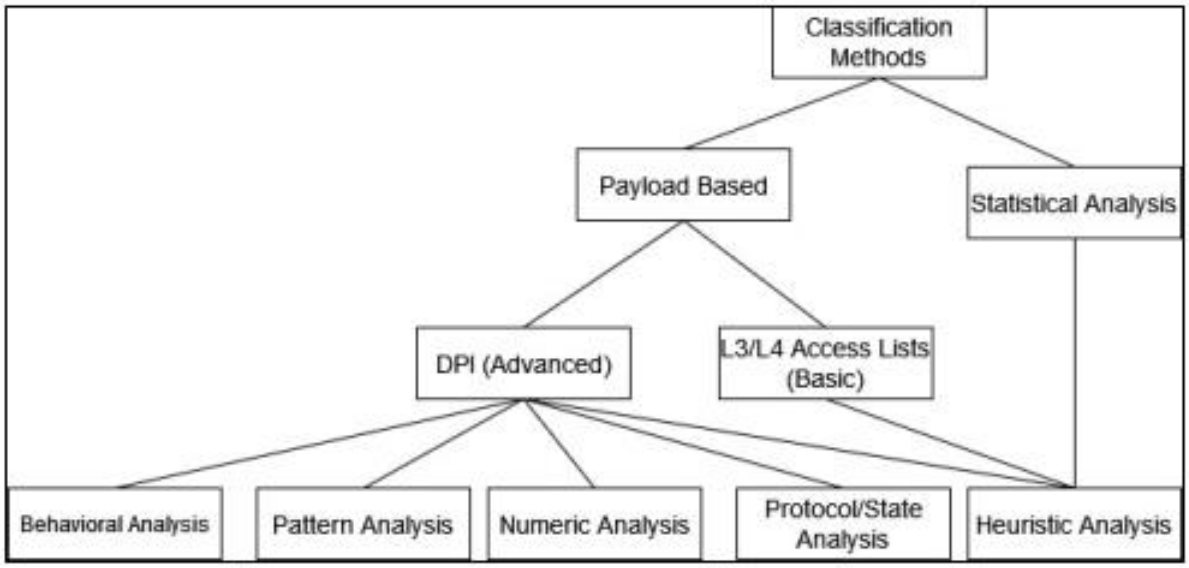

Рисунок 1.10 – Методи класифікації трафіку

Однак, цей підхід вважається застарілим і не дозволяє ефективно класифікувати дані більшості сучасних застосунків. Недолік полягає в тому, що багато сучасних програм не використовують стандартні порти для передачі інформації, а деякі програми обфускують свій трафік, маскуючись під інші. Таким чином, класифікація трафіку за допомогою базового підходу може бути недостовірною.

Більш досконалим підходом для класифікації є технологія глибокої інспекції пакетів (Deep Packet Inspection - DPI). Більшість рішень на базі DPI пропонують підхід аналізу сигнатур для ідентифікації та верифікації різних застосунків. Сигнатури – це унікальні "відбитки", які відповідають конкретній програмі. Отже, для кожного застосунку формується унікальна сигнатура, що характеризує його. З таких сигнатур створюється база даних. Класифікуюча програма потім порівнює отриманий трафік із цією базою даних і визначає, до якого застосунку належить даний трафік [4].

Аналізатор пакетів (також відомий як сніфер пакетів) – це програмне забезпечення або фізичне обладнання, що працює в комп'ютерній мережі та пасивно отримує дані канального рівня мережі у вигляді фреймів. Аналізатор пакетів перехоплює дані, які були адресовані іншим пристроям, зберігаючи їх для

подальшого аналізу.

Це програмне забезпечення може бути використане мережевими адміністраторами для моніторингу та діагностики помилок мережевого трафіку. Використовуючи інформацію, перехоплену аналізатором, адміністратор може виявити аномальні пакети, використовувати отриману інформацію для визначення вразливих місць у мережі та підтримувати ефективну роботу мережі.

1.7 Висновки до розділу

Проведений аналіз літературних джерел у межах дослідження предметної області показав, що розробка сучасних систем виявлення вторгнень із застосуванням алгоритмів машинного навчання є надзвичайно актуальним напрямом. Оглянуті наукові праці переконливо демонструють високу ефективність використання методів машинного навчання для завдань аналізу інтернет-трафіку. Це підтверджує значний потенціал та необхідність подальших досліджень у цій галузі.

2 СПЕЦІАЛЬНИЙ РОЗДІЛ

2.1 Технічні вимоги до програмного застосунку

2.1.1 Вимоги до етапів розробки та реалізації додатка для нейромережевого аналізу структури інтернет-трафіку

Задля створення ефективного застосунку для аналізу мережевого трафіку в режимі реального часу на основі методів машинного навчання, а також забезпечення його функціональності та відповідності визначеній меті, необхідно послідовно та ретельно виконати такі етапи:

– забезпечити проведення ґрунтовного аналізу предметної області: Це передбачає всебічне дослідження існуючих методів аналізу мережевого трафіку, виявлення актуальних проблем та викликів у сфері кібербезпеки та управління мережами, а також огляд сучасних досягнень у галузі машинного навчання, зокрема нейронних мереж, що можуть бути застосовані для класифікації мережевої активності;

– забезпечити збір та підготовку репрезентативного набору даних: Це включає визначення джерел мережевого трафіку (наприклад, публічні датасети, власні мережеві перехоплення), розробку методології збору даних, їх попередню обробку (очищення, нормалізація, вилучення ознак) та маркування для подальшого навчання моделей машинного навчання;

– забезпечити вибір оптимальних моделей машинного навчання: На основі проведеного аналізу предметної області та властивостей зібраного набору даних, необхідно обґрунтувати та обрати найбільш підходящі архітектури нейронних мереж або інші алгоритми машинного навчання, здатні ефективно вирішувати задачі класифікації мережевої активності;

– забезпечити реалізацію обраних моделей машинного навчання: Цей етап передбачає програмне кодування та конфігурацію вибраних моделей, їх навчання на підготовленому наборі даних, а також оптимізацію гіперпараметрів для досягнення максимальної продуктивності та точності класифікації;

– забезпечити проектування системи аналізу трафіку в реальному часі: Необхідно розробити архітектуру додатка, яка б дозволяла здійснювати безперервний збір мережевого трафіку, його оперативну обробку та подачу на вхід навчених моделей машинного навчання з мінімальною затримкою, враховуючи вимоги до швидкодії та масштабованості;

– забезпечити реалізацію системи аналізу інтернет-трафіку в реальному часі: Це включає створення програмного коду для всіх компонентів спроектованої системи: модуля збору трафіку, модуля попередньої обробки, модуля інтеграції з навченими моделями машинного навчання, а також інтерфейсів для відображення результатів аналізу та взаємодії з користувачем.

2.1.2 Вимоги до діагностування додатку для нейромережевого аналізу структури інтернет-трафіку

Для ефективного виявлення проблем у функціонуванні додатку для нейромережевого аналізу структури інтернет-трафіку та визначення їх типу, адміністратор системи повинен мати можливість переглядати поточні налаштування всіх компонентів, включаючи модулі збору даних, обробки, моделі машинного навчання та інтерфейси.

Якщо виявлено проблеми, пов'язані з некоректною роботою аналізу, рекомендується використання діагностичних інструментів та утиліт, вбудованих у систему, для перевірки цілісності та коректності надходження мережевого трафіку, а також для контролю за станом моделей машинного навчання. Це може включати моніторинг показників якості класифікації (точність, повнота, F1-міра), часу відгуку, завантаження ресурсів та інших внутрішніх метрик системи.

Особливу увагу слід приділити системам моніторингу трафіку, що є джерелом даних для аналізу. Для виявлення аномалій у вхідному трафіку та можливих проблем у мережевому з'єднанні, що можуть впливати на якість аналізу, рекомендується застосування таких інструментів, як ping або tracert для перевірки доступності мережевих джерел. Додатково, для перегляду та діагностики роботи з мережевим

трафіком, на відповідних пристроях чи серверах повинні бути налаштовані лінії VTY з протоколом доступу SSH для безпечного віддаленого управління та перегляду логів. Це дозволить оперативно ідентифікувати невідповідності у вхідних даних або збої у роботі компонентів, що відповідають за збір та попередню обробку трафіку.

2.1.3 Вимоги до показників призначення

Застосунок для нейромережевого аналізу структури інтернет-трафіку повинен забезпечувати наступні показники призначення:

– здатність ідентифікувати незвичайні патерни або відхилення від нормальної поведінки трафіку, що можуть свідчити про потенційні кіберзагрози або технічні збої;

– можливість точно розрізняти різні види трафіку, такі як веб-серфінг, потокове відео, голосовий зв'язок (voip), передача файлів, ігрова активність, а також службовий трафік;

– здатність визначати конкретні застосунки, які є джерелом або приймачем мережевого трафіку, навіть при використанні нестандартних портів або обфускації;

– функціональність для диференціації нормального мережевого потоку від трафіку, пов'язаного з атаками (наприклад, DDoS, сканування портів, шкідливе ПЗ) або несанкціонованою діяльністю;

– забезпечення безперервного аналізу та відображення динамічних змін у характеристиках мережевого трафіку, що дозволяє оперативно реагувати на нові загрози або зміни у поведінці мережі;

– надання інструментів для генерування детальних звітів щодо виявлених аномалій, класифікованого трафіку та ідентифікованих застосунків, а також їх графічного відображення для зручності інтерпретації;

– здатність системи до оновлення та перенавчання моделей для ефективної роботи з еволюціонуючими видами трафіку та новими кіберзагрозами.

До складу системи аналізу трафіку повинно входити шість основних функціональних модулів з визначеними ролями: модуль збору даних (отримання

трафіку з мережевих інтерфейсів), модуль попередньої обробки (нормалізація, вилучення ознак), модуль нейромережевої класифікації (застосування навчених моделей), модуль виявлення аномалій (ідентифікація відхилень), модуль візуалізації та звітності (інтерфейс користувача), та модуль управління моделями (оновлення та перенавчання).

2.2 Функціональні та нефункціональні вимоги додатку для нейромережевого аналізу структури інтернет-трафіку

Функціональні вимоги – це детальний перелік функцій або сервісів, які система зобов'язана виконувати, а також обмеження, що стосуються даних та поведінки системи [8].

Розроблюваний застосунок має відповідати таким функціональним вимогам. Користувач повинен мати можливість обирати попередньо навчену модель для проведення аналізу мережевого трафіку.

Застосунок має відображати інформацію про результати класифікації трафіку, а також повідомляти про будь-які помилки, що виникли під час роботи, для інформування користувача.

Застосунок повинен вести детальний журнал роботи, фіксуючи ключові події, час виконання операцій та виявлені аномалії для подальшого аналізу та діагностики.

Користувач повинен мати можливість обирати конкретний мережевий інтерфейс для проведення аналізу трафіку в режимі реального часу.

Користувач повинен мати можливість зберігати отримані записи (журнали, результати класифікації) про роботу застосунку для подальшого використання або архівування.

Користувач повинен мати можливість запускати та зупиняти процес аналізу мережевого трафіку за власним бажанням.

Нефункціональні вимоги визначають умови та середовище, в якому функціонують системні можливості (наприклад, аспекти захисту та доступу до баз даних, конфіденційність тощо). Вони не пов'язані безпосередньо з основними

функціями, але відображають очікування користувача щодо їх виконання. Ці вимоги характеризують принципи взаємодії системи з її оточенням або іншими системами, а також враховують аспекти продуктивності (час роботи), захисту даних та стандарти якості для досягнення конкретних показників чи атрибутів якості [9].

Розроблюваний застосунок має відповідати таким нефункціональним вимогам. Застосунок повинен бути розроблений з використанням мови програмування Python.

У застосунку повинно бути реалізовано щонайменше п'ять алгоритмів машинного навчання, які забезпечують різноманітні підходи до класифікації та аналізу трафіку.

Інтерфейс користувача повинен бути графічним (GUI) для забезпечення зручності та інтуїтивності взаємодії.

2.3 Сценарії використання системи

Діаграма варіантів використання – це графічне представлення, що ілюструє взаємозв'язки між акторами та варіантами використання системи. Створення такої діаграми має на меті досягнення кількох цілей: на початкових етапах проєктування системи визначити загальні межі та контекст моделюваної предметної області; сформулювати загальні вимоги до функціональної поведінки системи, що проєктується; розробити початкову концептуальну модель системи для її подальшої деталізації у вигляді логічних та фізичних моделей; а також підготувати вихідну документацію для ефективної взаємодії між розробниками системи, її замовниками та кінцевими користувачами. Сам варіант використання являє собою специфікацію загальних особливостей поведінки або функціонування моделюваної системи, без заглиблення у її внутрішню структуру.

На представленій діаграмі головним актором є "Користувач", який взаємодіє із системою. Користувач має один основний варіант використання системи – "Запустити аналіз трафіку" (рис.2.1). Для того, щоб ініціювати аналіз трафіку, користувачеві необхідно обрати попередньо навчену модель, а також визначити режим роботи. У застосунку передбачена робота у двох режимах: режим реального

часу та режим аналізу локальних файлів. Залежно від обраного режиму, користувач має або вибрати мережевий інтерфейс, дані з якого будуть прослуховуватися, або вказати файл, що підлягатиме аналізу. Після запуску аналізу трафіку користувач може припинити процес роботи застосунку, а також зберегти файл, який містить інформацію про результати його функціонування.

Рисунок 2.1 – Діаграма варіантів використання

2.4 Розробка програмної частини

Для розробки застосунку було обрано основну мову програмування Python версії 3.12.0 [13]. Ця мова пропонує широкий спектр бібліотек, що значно спрощують роботу з даними та реалізацію завдань машинного навчання.

Середовищем для створення та навчання моделей машинного навчання було обрано Google Colaboratory. Цей хмарний сервіс надає безоплатний доступ до обчислювальних ресурсів графічних (GPU) та тензорних (TPU) процесорів і функціонує на базі Jupyter Notebook. Google Colaboratory дозволяє виконувати код в окремих комірках, що прискорює налагодження програмного забезпечення, що розробляється. Крім того, Google Colaboratory вже містить попередньо встановлені популярні бібліотеки для роботи з даними та машинним навчанням.

Для реалізації моделей машинного навчання використовувалися бібліотеки scikit-learn, catboost та PyTorch Lightning. Бібліотека scikit-learn пропонує готові рішення для алгоритмів машинного навчання, набори даних, а також методи для обробки даних та аналізу результатів моделювання. PyTorch Lightning – це бібліотека для глибокого навчання, що базується на фреймворку PyTorch. Вона забезпечує гнучкий інтерфейс для розробки, навчання, тестування та аналізу навчених моделей

глибокого навчання, значно спрощуючи ці процеси.

Реалізація аналізатора пакетів виконана за допомогою бібліотеки Scapy. Scapy – це бібліотека, написана на Python, призначена для інтерактивної маніпуляції інтернет-пакетами.

Середовищем розробки було обрано програму Visual Studio Code [14].

2.4.1 Попередня обробка набору даних

Вихідні набори даних пройшли попередню обробку з метою підвищення ефективності роботи моделей. Цей процес складався з кількох послідовних етапів.

Спочатку було змінено назви ознак у використовуваних наборах даних. Також було усунено дублюючі ознаки, ознаки, що містили лише одне унікальне значення, та записи, які містили порожні значення.

Набори даних характеризувалися незбалансованою кількістю записів для кожного класу (рисунки 2.2–2.3). Для поліпшення якості моделей було проведено балансування для кожного набору даних. Класи, що містили занадто малу кількість записів, були відкинуті. Для кожного з решти класів було вибрано однакову кількість записів.

Під час балансування набору даних CIC-IDS-2017 було відібрано 5 499 записів для таких класів [11]:

- DoS Slowhttptest;
- DoS Slowloris;
- SSH-Patator;
- FTP-Patator;
- DoS GoldenEye;
- DDoS;
- PortScan;
- DoS Hulk;
- BENIGN (нормальний трафік).

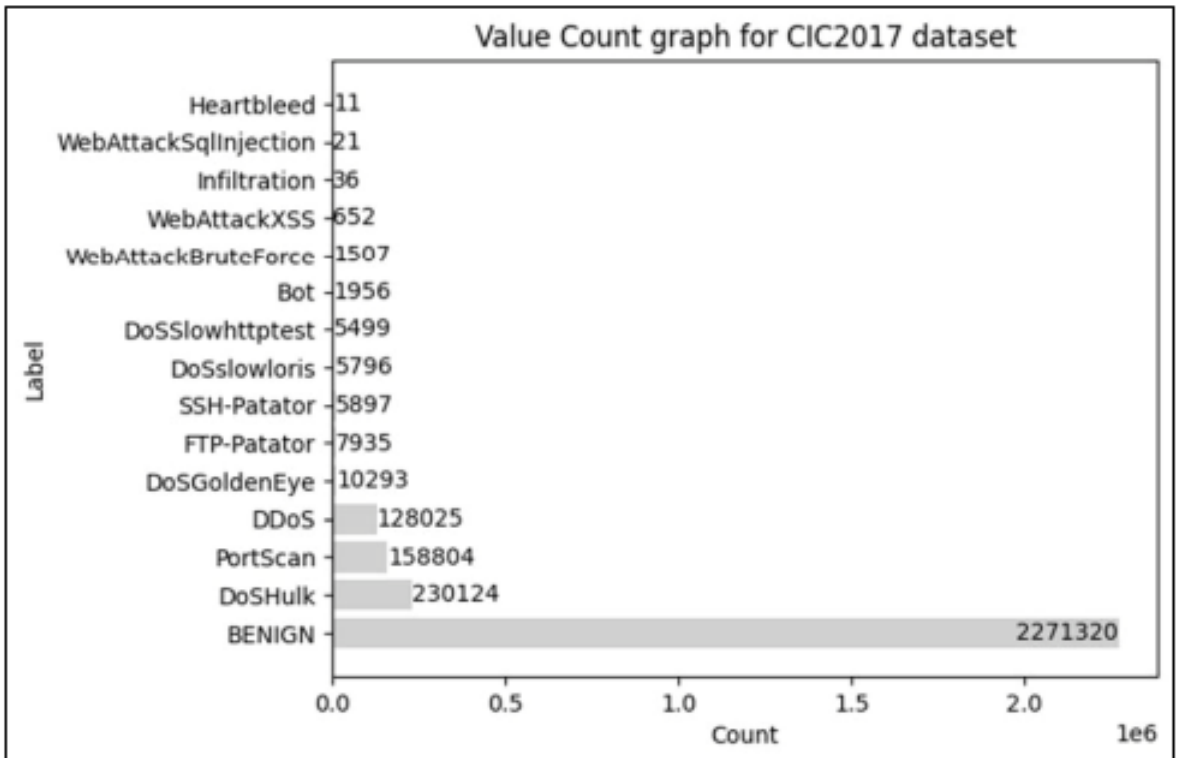

Рисунок 2.2 – Записи набору даних CIC-IDS-2017

За результатами балансування набору даних CSE-CIC-IDS2018 було відібрано 10 990 записів для таких класів [12]:

- DDoS attacks-LOIC-HTTP;
- DoS attacks-SlowHTTPTest;
- DoS attacks-Hulk;
- SSH-BruteForce;
- FTP-BruteForce;
- DoS attacks-Slowloris;
- DoS attacks-GoldenEye;
- Bot;
- DDOS attack-HOIC;
- Infiltration;
- Benign (нормальний трафік).

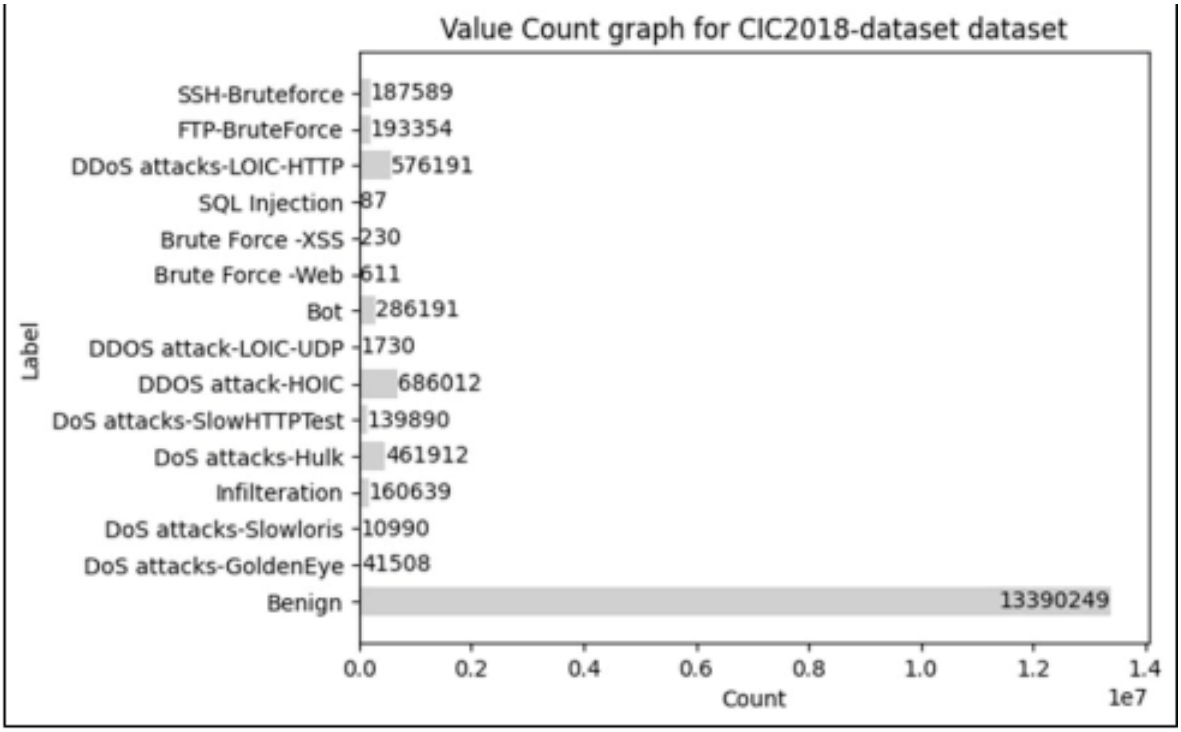

Рисунок 2.3 – Записи набору даних CSE-CIC-IDS2018

Дані в наборі даних були оброблені за допомогою методу StandardScaler, реалізованого в бібліотеці sklearn. Цей метод масштабує дані таким чином, щоб їх середнє значення дорівнювало нулю, а стандартне відхилення — одиниці.

Подальша попередня обробка полягала у відборі найбільш значущих ознак. Оскільки використовувані набори даних мають спільний набір ознак, відбір проводився з використанням набору даних CIC-IDS-2017. Попередньо, ваги ознак були отримані в ході навчання моделі RandomForestClassifier. На основі отриманих результатів було зроблено висновок про присутність у наборі ознак, які мінімально впливають на результат класифікації.

Для відбору ознак використовувався метод Recursive Feature Elimination з бібліотеки sklearn. Цей метод приймає на вхід модель-оцінювач та бажану кількість ознак, які необхідно залишити у вихідному наборі даних. На початку відбору модель-оцінювач навчається на повному наборі ознак. На кожній наступній ітерації визначаються ваги ознак, найменш важливі ознаки відкидаються, а оцінювач повторно навчається на зменшеному наборі ознак. Код відбору ознак представлений у лістингу 1 (рис.2.4).

```
def select_features(train_x, train_y):
  model = RandomForestClassifier(25, random_state=RANDOM_STATE)
  rfe = RFE(estimator = model, n_features_to_select=38)
  rfe.fit(train_x, train_y)
  feature_map = [(i, v) for i, v in itertools.zip_longest(rfe.get_support(),
train_x.columns)]
  selected_features = [v for i, v in feature_map if i==True]
 return pd.DataFrame(rfe.support_,index=train_x.columns,columns=["Rank"]),
selected_features
```

Рисунок 2.4 – Лістінг 1

Як видно з лістингу, функція повертає таблицю, що містить інформацію про використання ознак, та список відібраних ознак. Відібрані ознаки представлені на рисунку 2.5.

```
import pprint
pprint.pprint(olo, width=60, compact=True)
```

```
['dst_port', 'flow_duration', 'tot_fwd_pkts',
 'tot_bwd_pkts', 'totlen_fwd_pkts', 'totlen_bwd_pkts',
 'fwd_pkt_len_max', 'fwd_pkt_len_mean', 'fwd_pkt_len_std',
 'bwd_pkt_len_max', 'bwd_pkt_len_min', 'flow_byts/s',
 'flow_pkts/s', 'flow_iat_mean', 'flow_iat_std',
 'flow_iat_max', 'fwd_iat_tot', 'fwd_iat_mean',
 'fwd_iat_max', 'fwd_header_len', 'bwd_header_len',
 'bwd_pkts/s', 'pkt_len_max', 'pkt_len_mean', 'pkt_len_std',
 'pkt_len_var', 'psh_flag_cnt', 'pkt_size_avg',
 'fwd_seg_size_avg', 'bwd_seg_size_avg', 'subflow_fwd_pkts',
 'subflow_fwd_byts', 'subflow_bwd_byts',
 'init_fwd_win_byts', 'init_bwd_win_byts',
 'fwd_seg_size_min', 'active_min', 'idle_mean']
```

Рисунок 2.5 – Проаналізовані ознаки

Отже, в ході попередньої обробки вихідні набори даних були уніфіковані, було відкинуто малозначущі ознаки та ознаки з єдиними значеннями, а також проведено балансування наборів даних.

2.4.2 Реалізація алгоритму машинного навчання

Рекурентна нейронна мережа (РНМ) – це архітектура штучних нейронних мереж, спеціально адаптована для роботи з послідовними даними, такими як часові ряди. Такі мережі призначені для вирішення завдань, у яких вхідні дані залежать від попередніх даних, що вимагає збереження інформації про попередні стани мережі [10].

В архітектурі такої нейронної мережі кожен вузол отримує на вхід стан з попереднього вузла та передає результат своєї роботи на наступну ланку. Приклад архітектури такої мережі представлений на рисунку 2.6.

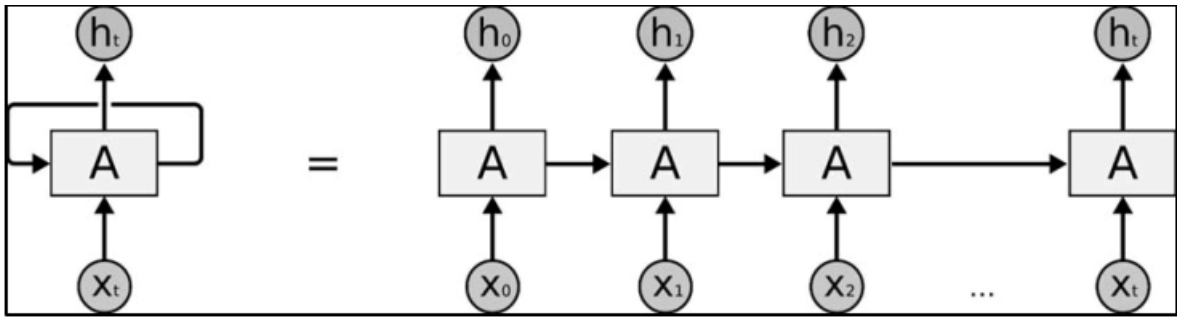

Рисунок 2.6 – Схема рекурентної мережі

Однак у таких нейронних мережах контекст попередньої інформації швидко зникає. Тому для завдань, де для прийняття рішень потрібна більше інформації з попередніх станів, така мережа не підходить.

Для вирішення цієї проблеми була запропонована архітектура Long Short-Term Memory (Довга Короткочасна Пам'ять). Long Short-Term Memory (LSTM) – це підклас рекурентних нейронних мереж (РНМ), здатних навчатися довгостроковим залежностям. Приклад архітектури LSTM представлений на рисунку 2.7.

Основною відмінністю LSTM мережі від РНМ є будова вузла. У РНМ-мережах для отримання результату для поточного вузла вихідні дані попереднього вузла додаються до вхідних даних поточного вузла, після чого до даних застосовується функція активації tanh. У LSTM модуль складається з декількох частин. Основна складова модуля – це лінія стану вузла. Інформація надходить на цю лінію з різних шарів мережі.

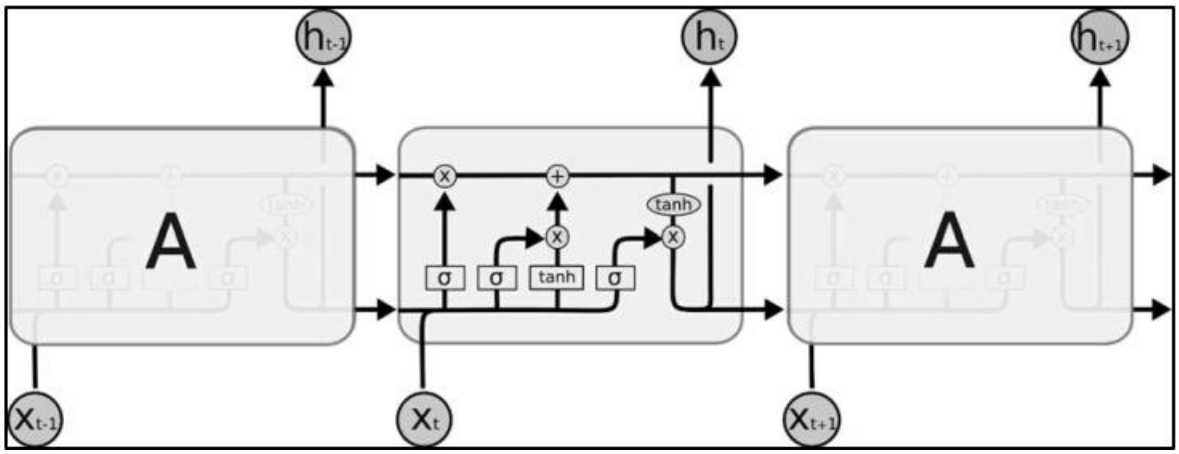

Рисунок 2.7 – Схема LST- мережі

На початку роботи LSTM-вузла відбувається конкатенація поданих на вхід поточної комірки даних та вихідного значення з минулого стану. Перший шар мережі – шар фільтра забування. На цьому шарі за допомогою сигмоїдної функції активації визначається, яка інформація буде відкинута: значення 1 означатиме, що дані потрібно залишити; значення 0 означатиме, що дані потрібно забути.

Наступний шар визначає інформацію, яка буде додана до стану вузла. Цей шар складається з двох етапів. На першому етапі за допомогою сигмоїдної функції активації визначаються оновлювані значення. На другому етапі за допомогою функції активації tanh створюється вектор значень-кандидатів, які можуть бути додані до лінії стану. Після чого дані, що пройшли через сигмоїдну та tanh функції активації, комбінуються для оновлення стану комірки.

Наступний шар визначає вихідну інформацію комірки. Для отримання вихідної інформації дані з лінії стану вузла, що пройшли через функцію активації tanh, комбінуються з вхідними даними, що пройшли через сигмоїдну функцію активації, отримуючи вихідний сигнал комірки та стан для наступного модуля LSTM мережі.

Таким чином, модель LSTM дозволяє зберігати необхідний для завдань контекст довше та долати проблему довгострокових залежностей, досягаючи кращих результатів, ніж моделі, засновані на PHM-архітектурі [10].

Для реалізації моделі LSTM (Long Short-Term Memory) було створено клас LSTMModel, який визначає архітектуру нейронної мережі та реалізує навчальний і тестовий кроки. Нейронна мережа складається з шару LSTM, функції активації ReLU

та повнозв'язного шару. Код моделі представлено в Лістингу 2 (рис.2.8).

```
class LSTMModel(pl.LightningModule):
  def __init__(self, input_size, hidden_size, num_layers, num_classes, lr):
  super().__init__()
  self.hidden_size = hidden_size
  self.num_layers = num_layers
  self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
  self.fc = nn.Linear(hidden_size, num_classes)
  self.relu = nn.ReLU()
  self.learning_rate = lr

  def forward(self, x):
 h0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hid-
den_size).to(self.device))
 c0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hid-
den_size).to(self.device))
   out, (hn, cn) = self.lstm(x, (h0, c0))
   hn = hn.view(-1, self.hidden_size)
   out = self.relu(hn)
   out = self.fc(out)
   return out

  def training_step(self, batch, batch_idx):
   x, y = batch
   y_hat = self(x)
   loss = F.cross_entropy(y_hat, y)
   self.log('train_loss', loss, on_step=False, on_epoch=True, prog_bar=True)
   return loss

def validation_step(self, batch, batch_idx):
   x, y = batch
   y_hat = self(x)
   loss = F.cross_entropy(y_hat, y)
   self.log('val_loss', loss, on_step=False, on_epoch=True, prog_bar=True)
   return loss

  def test_step(self, batch, batch_idx):
   x, y = batch
   y_hat = self(x)
   loss = F.cross_entropy(y_hat, y)
   self.log('test_loss', loss)
   return loss
```

Рисунок 2.8 – Лістінг 2

Для ефективної роботи з даними було реалізовано клас CICIDSDataset. Цей клас призначений для зберігання отриманих даних у форматі DataFrame та містить перевизначені методи __len__ (для отримання розміру набору даних) та __getitem__ (для отримання попередньо обробленого запису за індексом). Код класу представлений у Лістингу 3 (рис.2.9).

```
class CICIDSDataset(Dataset):
  def __init__(self, X: pd.DataFrame, y: pd.DataFrame):
   self.X = X
   self.y = y

  def __len__(self):
   return len(self.X)

  def __getitem__(self, idx):
   return torch.tensor(self.X.iloc[idx], dtype=torch.float32).unsqueeze(0),
torch.tensor(self.y[idx], dtype=torch.int64)
```

Рисунок 2.9 – Лістінг 3

Ансамблеве навчання – це техніка машинного навчання, яка збільшує точність та достовірність передбачень моделі шляхом об'єднання передбачень декількох моделей. Мета такого підходу – зменшити помилки або зміщення, які можуть бути присутніми в окремих моделях. Враховуючи різні слабкі та сильні сторони простих моделей, покращується загальна продуктивність навченої системи. Також такий підхід дозволяє справлятися з різними похибками даних

Stacking (Стек/Комбінування). У цій категорії використовуються передбачення кількох різних моделей для побудови нової моделі. Нова модель використовується для передбачень на тестовому наборі даних. Суть методу можна описати так: з кожної моделі, що використовується в ансамблі, на основі навчальної вибірки створюється набір ознак, на основі отриманих ознак навчається підсумкова ансамблева модель.

Bagging (Беггінг) або ж Bootstrap Aggregation (Агрегування за допомогою бутстрепу). У цій категорії методів безліч базових моделей незалежно та паралельно навчаються на унікальних вибірках вихідного набору даних, отриманих за допомогою техніки Bootstrap. Результат передбачення визначається голосуванням. Bootstrap – це технологія семплінгу, в якій створюються вибірки оригінального набору даних з повтореннями. Розмір отриманих вибірок відповідає розміру оригінального набору даних . Суть методу можна описати так: генерація вибірок вихідного набору розміром B з вихідного набору даних розміру N здійснюється шляхом випадкового вибору з повтореннями елементів у кожному з B разів .

Boosting (Бустинг). Boosting – це послідовна техніка навчання ансамблевих моделей, в якій кожна наступна базова модель навчається на помилках попередньої. Алгоритм навчання можна описати наступним чином: на кожному кроці навчання базової моделі збільшуються ваги некоректно класифікованих записів вихідного набору даних. Наступна модель намагається виправити помилки попередньої моделі. Фінальна модель є зваженим середнім усіх базових моделей. Ідея методу полягає в тому, що базові моделі погано працюють на загальному наборі даних, але можуть добре справлятися з частиною набору; об'єднавши декілька базових моделей, можливо збільшити загальну ефективність ансамблевої моделі."

Для керування процесом навчання моделі був використаний об'єкт класу Trainer з бібліотеки pytorch-lightning. Цей клас відповідає за управління всіма етапами навчання, валідації та тестування моделей, значно спрощуючи керування життєвим циклом моделі. Код навчання моделі представлений у Лістингу 4 (рис.2.10).

```
|
input_size = 38
hidden_size = 128
num_layers = 1
num_classes = len(l_encoder.classes_)
model = LSTMModel(input_size, hidden_size, num_layers, num_classes, 1e-4)
train_dataset = CICIDSDataset(train_x, train_y)
train_dataloader = DataLoader(train_dataset, batch_size=32, shuffle=True)
trainer = pl.Trainer(max_epochs=50, callbacks=[pl.callbacks.EarlyStop-
ping(monitor="train_loss", mode="min")])
trainer.fit(model, train_dataloader)
```

Рисунок 2.10 – Лістінг 4

Метод опорних векторів (SVM) є одним із класичних алгоритмів машинного навчання, який застосовується для розв'язання задач класифікації та регресії. Основна ідея методу полягає в побудові гіперплощини, що оптимальним чином розділяє об'єкти вибірки. Алгоритм працює з припущення, що чим більша відстань між розділяючою гіперплощиною та об'єктами розділюваних класів, тим меншою буде середня помилка класифікатора.

Гіперплощина – це межа рішень, яка розділяє дані на різні класи в

багатовимірному просторі. У N-вимірному просторі гіперплощина має N−1 вимірність. За допомогою гіперплощини дані розділяються на два класи, опинившись по один або інший бік площини.

Зазор – це відстань між гіперплощиною та найближчими точками даних для кожного класу. Зазор є мірою вимірювання якості розділення класів у просторі ознак.

Опорні вектори – це дані, що знаходяться найближче до межі рішень. Ці точки є важливими, тому що вони визначають позицію та напрямок гіперплощини, а також мають велике значення для точності класифікації.

Алгоритм SVM визначає оптимальну гіперплощину, яка максимізує зазор між класами. Алгоритм SVM може працювати як з лінійно розділюваними даними, так і з нелінійно розділюваними, проєктуючи дані у простір більшої розмірності, де класи стають розділюваними гіперплощиною.

CatBoost (Categorial Boosting) – це бібліотека з відкритим вихідним кодом, що реалізує алгоритм градієнтного бустингу на основі дерев рішень. Бібліотека розроблена компанією Яндекс. Перша версія бібліотеки була представлена у 2017 році. Наразі бібліотека використовується в проєктах Яндекса, пов'язаних з машинним навчанням, а також у багатьох інших компаніях, серед них: CERN, Cloudflare, Careem taxi.

Серед переваг бібліотеки відзначають: відсутність необхідності в підборі параметрів; підтримку категоріальних ознак; наявність версії для GPU; високу точність; швидкий час роботи.

Підвищення точності моделей досягається за допомогою обробки категоріальних ознак та комбінування ознак. Обробка категоріальних ознак полягає в заміні категорій середнім значенням для класу на випадкових змінених вибірках даних. Комбінування ознак дозволяє об'єднувати кілька категоріальних ознак в одну, що дозволяє більш ефективно обробляти категоріальні ознаки та втрачати менше інформації .

2.4.3 Оцінка навчання моделі машинного навчання

Для цілей навчання та тестування, вихідний набір даних було розділено на дві частини: навчальну та тестову. Навчальна частина становить 80% від загального обсягу даних, тоді як тестова частина складає 20%.

За результатами навчання можливо розрахувати метрики якості моделей. Для оцінки ефективності навчених моделей використовувалися такі метрики: точність (Accuracy), прецизія (Precision), повнота (Recall) та F1-міра (F1-score).

Accuracy – це метрика, яка відображає відношення кількості правильно спрогнозованих записів до загальної кількості всіх спрогнозованих записів.

Precision – метрика, що обчислюється як відношення кількості істинно позитивних результатів до загальної кількості всіх спрогнозованих позитивних результатів.

Recall (Повнота) – це співвідношення істинно позитивно спрогнозованих записів у певному класі до загальної кількості всіх правильно спрогнозованих записів (які належать до цього класу).

F1-score – є гармонійним середнім між Precision та Recall; високі значення цієї метрики свідчать про високі значення обох показників – Precision та Recall.

На основі обчислення метрик оцінки якості було сформовано звіт про навчання. Цей звіт представлено на рисунках 2.11–2.12. Виходячи з аналізу цих рисунків, зроблено висновок, що всі навчені моделі продемонстрували високу точність на тестовій вибірці. Найкращий результат за метрикою F1-score досягла модель Random Forest, навчена на наборі даних CIC-IDS-2017.

	Model	Accuracy	Precision	Recall	F1 score
0	Random Forest	0.997980	0.997983	0.997980	0.997980
1	AdaBoost	0.995858	0.995911	0.995858	0.995857
2	SVM	0.961511	0.963758	0.961511	0.961587
3	CatBoost	0.997879	0.997881	0.997879	0.997878
4	LSTM	0.986766	0.987180	0.986766	0.986769

Рисунок 2.11 – Результат навчання на наборі даних CIC-IDS-2017

	Model	Accuracy	Precision	Recall	F1 score
0	Random Forest	0.894822	0.915810	0.894822	0.889139
1	AdaBoost	0.891306	0.894610	0.891306	0.889864
2	SVM	0.881256	0.910099	0.881256	0.872426
3	CatBoost	0.909463	0.915243	0.909463	0.907609
4	LSTM	0.893209	0.899780	0.893209	0.890937

Рисунок 2.12 – Результат навчання на наборі даних CSE-CIC-IDS2018

2.4.4 Реалізація аналізатора пакетів

Аналізатор пакетів був реалізований із застосуванням бібліотеки Scapy. Для отримання необроблених пакетів з користувацького інтерфейсу використовувався клас AsyncSniffer. Цей клас дозволяє асинхронно отримувати інтернет-пакети, а також ефективно керувати процесом їхнього збору.

Отримання інформації з пакета реалізовано за допомогою класу PacketInfo. Клас PacketInfo описує структуру зберігання параметрів пакетів і містить методи для встановлення та отримання різних параметрів пакета. Параметри встановлюються шляхом перетворення інформації з різних рівнів архітектури TCP/IP.

Для отримання та обробки інтернет-пакетів було розроблено клас Worker. Цей клас керує процесом отримання пакетів, містить методи для обробки вхідних та вихідних інтернет-пакетів, а також метод для їх класифікації.

Для обробки пакетів, отриманих за допомогою AsyncSniffer, був реалізований метод newPacket. У цьому методі описано процес вилучення інформації з необробленого пакета та реалізовано логіку управління потоками. Вихідний код методу newPacket представлений у Додатку А. У методі здійснюється перевірка належності пакета до існуючих потоків. Якщо пакет не належить до жодного потоку, на основі отриманого пакета створюється новий потік. У програмі визначено час бездіяльності потоку. У разі перевищення цього часу бездіяльності викликається метод класифікації, поточний потік знищується і створюється новий потік на основі останнього отриманого пакета. Якщо останній отриманий пакет містить прапорець закінчення потоку, також викликається метод класифікації і потік знищується.

При зупинці роботи AsyncSniffer для кожного потоку, що залишився активним, викликається метод класифікації.

Класифікація потоку реалізована в методі classify. У цьому методі відбувається масштабування отриманих даних, їх класифікація та виведення інформації в консоль та користувацький інтерфейс. Код методу представлений у Лістингу 5 (рис.2.13).

```
def classify(self, features):
  # preprocess
  try:
   f = features
   lv = 0
   features = self.normalization.transform([f])
   result = self.model.predict(features)

   feature_string = [str(i) for i in f]
   classification = [str(result[0])]
   result_string = f"[{datetime.now().strftime(self.format)}] - {classifica
tion[0]}"
   if result in ['BENIGN', 'Benign']:
    lv = 1

   self.signals.prints.emit([result_string, lv])
  except Exception as e:
   print(result)
   exctype, value = sys.exc_info()[:2]
   self.signals.error.emit((exctype, value, traceback.format_exc()))
  return feature_string + classification
```

Рисунок 2.13 – Лістінг 5

2.4.5 Реалізація користувальницького інтерфейсу

На основі розробленої діаграми варіантів використання було реалізовано користувацький інтерфейс застосунку. Для цього використовувалася бібліотека PyQT6 [15], яка є обгорткою для кросплатформного фреймворку Qt, призначеного для створення графічних інтерфейсів.

Головне вікно застосунку реалізовано за допомогою класу MainWindow, який успадковується від класу QMainWindow. Створення нових елементів у вікні програми здійснюється шляхом додавання відповідних віджетів. За допомогою цих віджетів

були додані такі елементи, як: кнопки, чекбокси та випадаючі списки.

Взаємодія з елементами інтерфейсу в бібліотеці PyQT6 обробляється за допомогою механізму сигналів і слотів. Сигнали – це дані, які віджет надсилає при взаємодії (наприклад, натискання кнопки). Слот – це функція, яка обробляє отриманий сигнал. Приклад обробки сигналу від чекбокса представлений у Лістингу 6 (рис.2.14).

```
self.workMode = QCheckBox('Use offline analysis')
self.workMode.setCheckState(Qt.CheckState.Unchecked)
self.workMode.stateChanged.connect(self.activateCheckBox)

def activateCheckBox(self, s):
  if s == 0:
  self.workModeLayout.setCurrentIndex(0)
  self.selectedWorkMode = "Live"
  if s == 2:
  self.workModeLayout.setCurrentIndex(1)
  self.selectedWorkMode = "Offline"
```

Рисунок 2.14 – Лістінг 6

Користувацький інтерфейс складається з таких елементів (рис.2.15): кнопка "Run Detection" (Запустити виявлення); випадаючий список "model" (модель); кнопка "Stop" (Зупинити); випадаючий список "interface" (інтерфейс); чекбокс "Use offline analysis" (Використовувати офлайн-аналіз); кнопка "Select File" (Вибрати файл); текстове поле; кнопка "Save Logs" (Зберегти логи).

Кнопка "Run Detection" відповідає за запуск процесу аналізу трафіку з обраними параметрами. При її натисканні надсилається сигнал, який викликає функцію runButton_was_clicked. У цій функції фіксується обрана модель, обраний мережевий інтерфейс або обраний файл для аналізу, а зафіксовані параметри передаються екземпляру класу Worker. Після цього відповідні кнопки стають неактивними, і за допомогою класу QThreadPool запускається процес аналізу трафіку. Вихідний код цієї функції представлений у Додатку А.

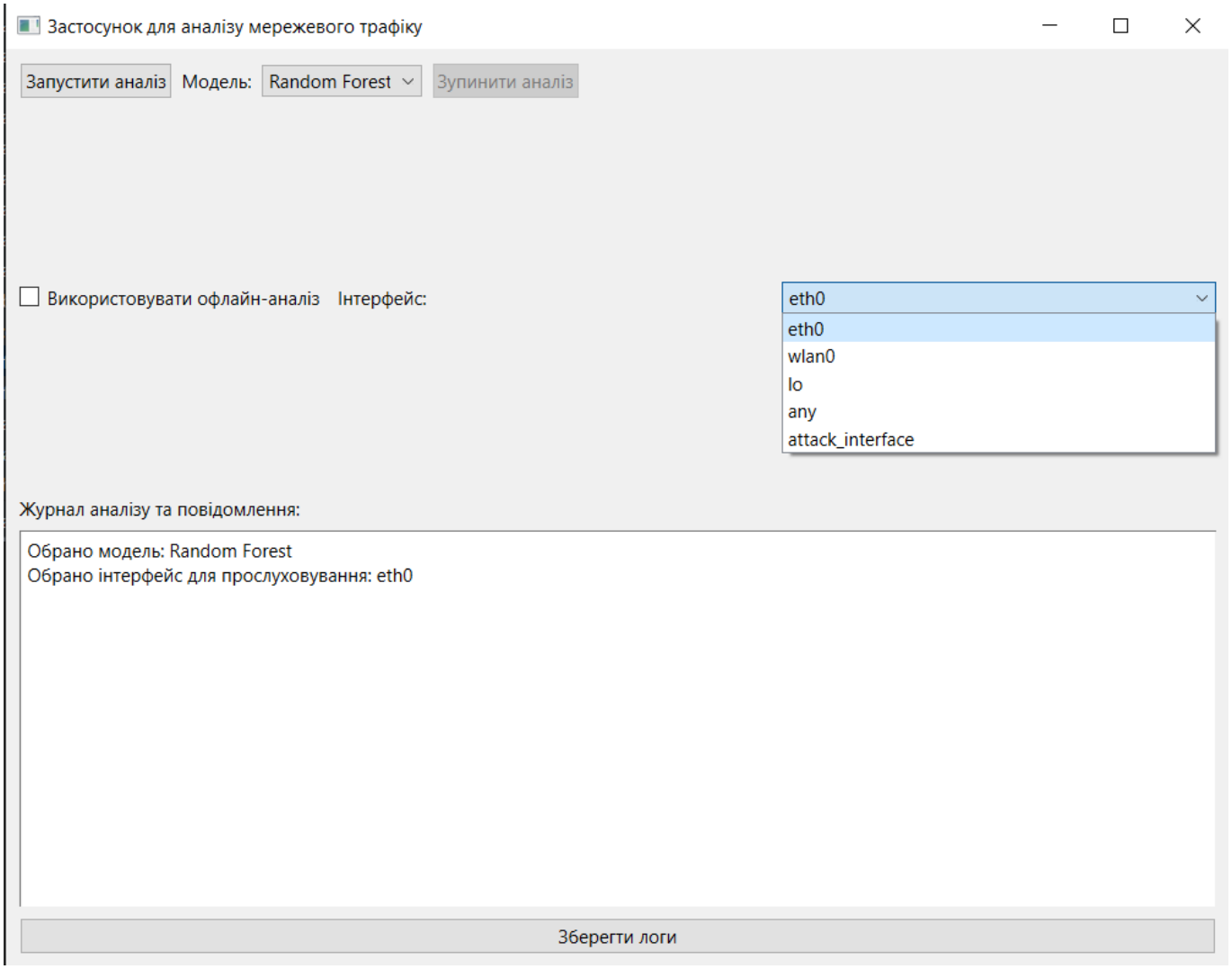

Рисунок 2.15 – Користувальницький інтерфейс

Кнопка "Stop" відповідає за зупинку процесу аналізу трафіку. При її натисканні в екземплярі класу Worker викликається метод stop. Виклик методу stop призводить до завершення роботи AsyncSniffer та надсилання сигналу про завершення роботи до MainWindow. Обробка цього сигналу відбувається у функції thread_complete. Ця функція обнуляє дані екземпляра класу Worker та розблоковує раніше заблоковані кнопки.

У випадаючому списку "model" користувач може обрати попередньо навчену модель машинного навчання, яка згодом буде використана у процесі аналізу трафіку.

У випадаючому списку "interface" користувач може обрати мережевий інтерфейс, який буде прослуховуватися під час аналізу трафіку.

Чекбокс "Use offline analysis" визначає режим роботи застосунку. При

активації цього чекбокса інтерфейс змінюється, і користувачеві пропонується обрати локальний файл у форматі pcap.

Кнопка "Select File" викликає діалогове вікно вибору файлу. Шлях до обраного файлу зберігається у локальну змінну, яка потім передається екземпляру класу Worker.

У текстовому полі відображаються результати роботи аналізатора трафіку, а також системні повідомлення про функціонування програми.

Кнопка "Save Logs" викликає функцію saveLogsButton_was_clicked. Ця функція отримує інформацію з текстового поля та зберігає її у файл з датою останнього запуску. При натисканні на кнопку текстове поле також очищається.

2.4.6 Тестування додатку

Користувач системи аналізу мережевого трафіку легко запускає аналіз за допомогою кнопки "Запустити виявлення", маючи можливість обрати попередньо навчену модель з випадаючого списку "Модель", що дозволяє адаптувати процес аналізу. Система пропонує два режими роботи: "Режим реального часу" для моніторингу живого трафіку з обраного мережевого інтерфейсу (наприклад, "eth0", "wlan0"), або "Офлайн-аналіз", коли користувач активує чекбокс "Використовувати офлайн-аналіз" та обирає файл у форматі .pcap. Під час роботи системи, у "Журналі аналізу та системних повідомленнях" відображаються хід процесу та системні повідомлення, такі як "Запуск нового аналізу" або "Аналіз офлайн. Порція даних". Користувач може зупинити аналіз у будь-який момент за допомогою кнопки "Зупинити". Результати класифікації трафіку та інші повідомлення виводяться в текстове поле журналу, надаючи миттєвий зворотний зв'язок. Усі отримані логи можна зберегти в текстовий файл, використовуючи кнопку "Зберегти логи", що дозволяє вести архів даних для подальшого використання. Після збереження текстове поле логів очищається, готуючи інтерфейс до нового аналізу (рис.2.16).

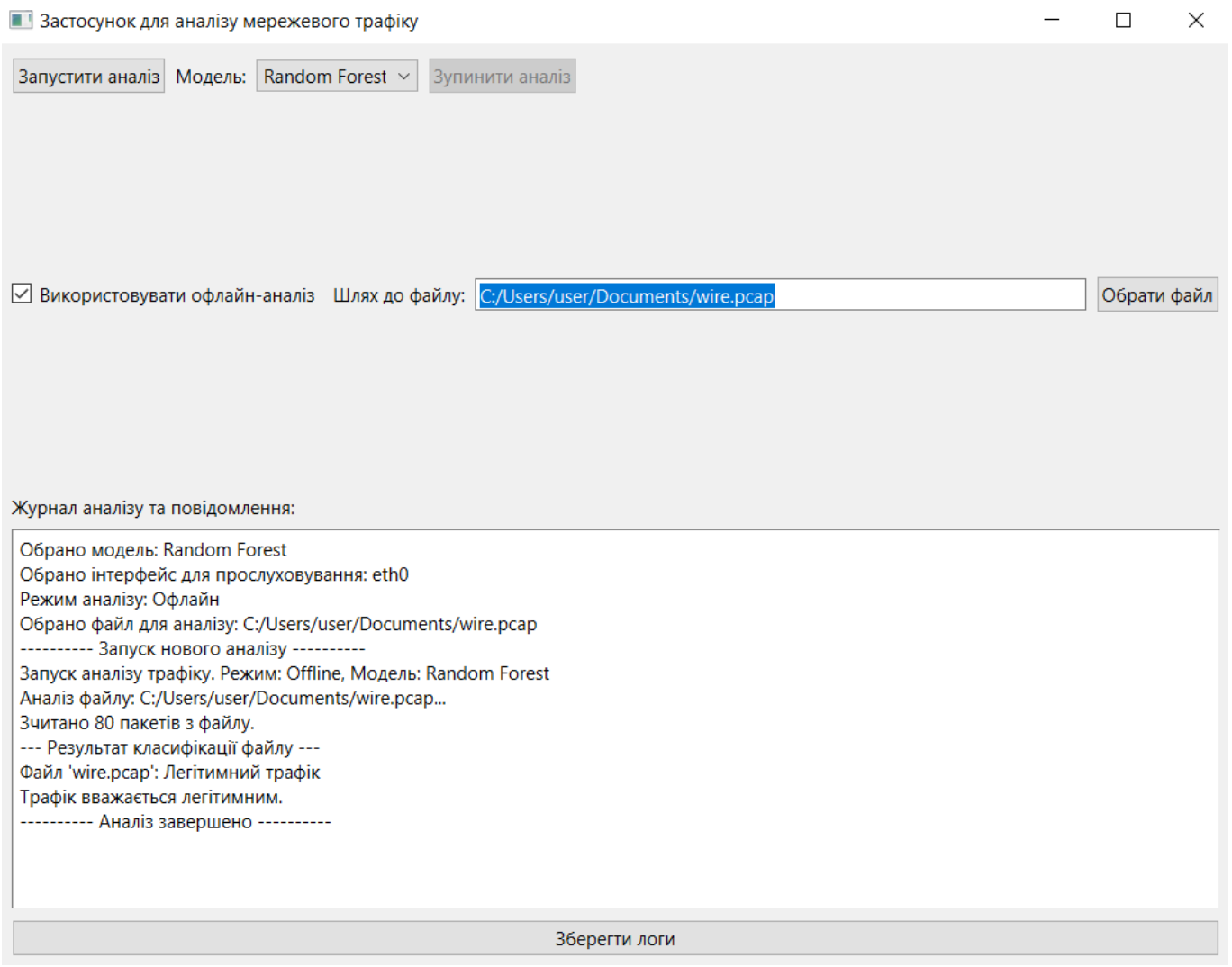

Рисунок 2.16 – Результат моделі Random Forest

Програмний засіб для аналізу мережевого трафіку дозволяє виявляти атаки в режимах реального часу та офлайн, використовуючи різні моделі машинного навчання, такі як Random Forest (рис.2.17).

У режимі реального часу, при виборі інтерфейсу attack_interface, система успішно ідентифікує DDoS (UDP Flood) атаки, про що свідчать багаторазові повідомлення "!!!!! ВИЯВЛЕНО ПІДОЗРІЛУ АКТИВНІСТЬ !!!!!". При цьому, між виявленнями атак, система також класифікує трафік як легітимний, показуючи "Поточна класифікація: Легітимний трафік".

В офлайн-режимі, шляхом завантаження файлу wire.pcap, програма виконує аналіз трафіку, імітуючи обробку даних порціями, та після завершення аналізу вказує, що аналіз завершено.

Загальна кількість атак у 2023 році становила 2 266 198, що є значним зростанням порівняно з 1 255 573 атаками у 2022 році. Розподіл атак за місяцями показує пік активності у листопаді 2023 року, коли кількість атак сягнула 339 502, тоді як у березні цього ж року було зафіксовано 173 759 атак. Аналогічно, інший показник загальної кількості атак у 2023 році склав 3 442 734, порівняно з 1 446 216 у 2022 році, з найбільшою кількістю атак у листопаді 2023 року – 299 882, та 153 288 у лютому.

Історично, найбільші DDoS-атаки за кількістю запитів на секунду (rps), зафіксовані Cloudflare, демонструють експоненціальне зростання: від 3 мільйонів rps у 2019 році до 201 мільйона rps у 2023 році.

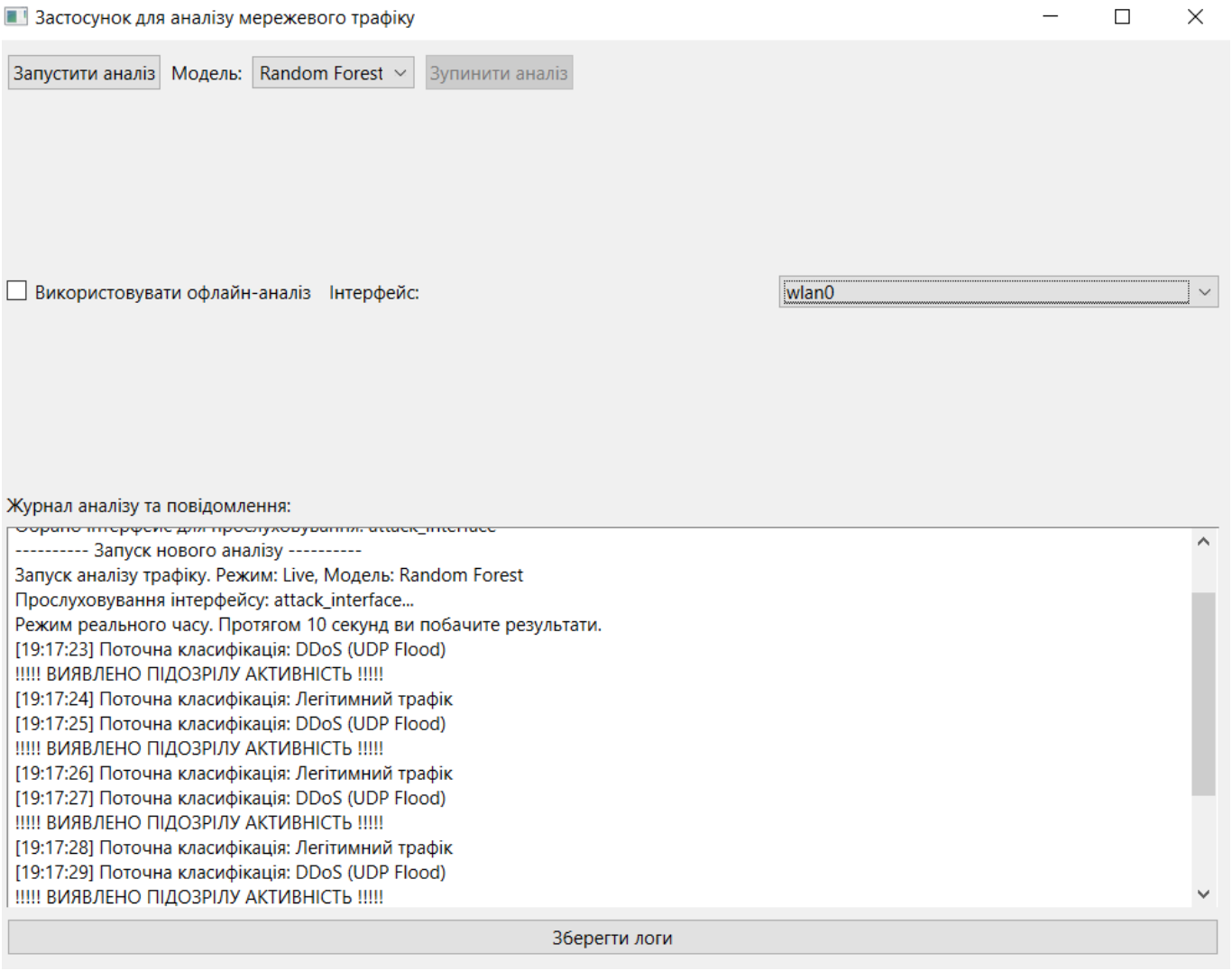

Рисунок 2.17 – Результат моделі Random Forest на виявлення атаки

Система аналізу трафіку, згідно з діаграмою варіантів використання, дозволяє користувачеві запускати аналіз трафіку, вибирати навчену модель, вибирати режим

роботи (реальний час або офлайн), вибирати інтерфейс для прослуховування або файл для аналізу, а також зупиняти аналіз трафіку та зберігати результати роботи.

Порівняння показників за кварталами (2020-2021) показує, що у Q1 2021 року загальна кількість атак становила 71.53%, тоді як у Q4 2020 року цей показник був 48.61%. При цьому відсоток "розумних атак" ("Smart attacks") у Q1 2021 склав 71.53%, у Q4 2020 – 57.41%, а у Q1 2020 – 88.89%, при загальному показнику 100% у Q1 2020.

Для обчислення статистичних характеристик та опису мережевих потоків були реалізовані класи Flow та FlowFeature. У класі Flow були реалізовані методи new та terminated. Метод new описує процес оновлення характеристик потоку при отриманні нового пакета. Метод terminated описує процес отримання фінальних характеристик перед завершенням та знищенням потоку. Характеристики мережевого потоку зберігаються у класі FlowFeature. Клас FlowFeature містить методи для встановлення та отримання цих характеристик потоку.

В ході даного тестування було перевірено відповідність системи заявленим функціональним вимогам. Для проведення тестування були створені тестові випадки. Результати тестування представлені в таблиці 2.1.

Таблиця 2.1 – Функціональне тестування

№	Назва тесту	Дії	Результат	Тест пройдено?
1	Вибір моделі	1. Натиснути на спадне меню з моделями.	В текстовому полі виводиться запис, що модель була обрана.	Так
2	Результат класифікації	Натиснути на кнопку «Запустити виявлення».	Додаток починає процес аналізу трафіку, в текстовому полі виводиться інформація про хід	Так

			роботи.	
3	Ведення журналу	1. Запустити додаток.	Додаток починає записувати інформацію про хід роботи.	Так
2. Натиснути на кнопку «Запустити виявлення».				
4	Зміна мережевого інтерфейсу	1. Натиснути на спадне меню з мережевим інтерфейсом.	У графічному інтерфейсі зміниться інтерфейс. В текстовому полі з'явиться запис про успішну зміну.	Так

В ході A/B тестування порівнюються два об'єкти – «як було» та «як стало» – з метою визначення впливу внесених змін на результат. У рамках машинного навчання таке тестування можна провести, варіюючи кількість тестових даних або змінюючи гіперпараметри.

Для обраних моделей було проведено A/B тестування, під час якого варіювалася кількість записів при балансуванні даних, а також оцінювалися результати на вихідних даних без балансування. Результати тестування моделей представлені в таблиці 2.2.

Таблиця 2.2 - A/B тестування

Кількість записів на клас	Random Forest	AdaBoost	SVM	CatBoost	LSTM
1 000	99,3%	98,7%	92,3%	99,5%	94,1%
2 000	99,6%	94,0%	94,5%	99,8%	97,3%
3 000	99,5%	99,5%	94,7%	99,7%	97,4%
5 499	99,8%	99,3%	95,4%	99,8%	98,5%
Без балансування	99,8%	98,9%	85,3%	99,8%	98,1%

2.5 Висновки до розділу

У другому розділі було детально розглянуто технічні, функціональні та нефункціональні вимоги до програмного застосунку для нейромережевого аналізу структури інтернет-трафіку, а також описано процес його розробки.

Початково, були сформульовані технічні вимоги, що охоплюють етапи розробки та реалізації додатка, включаючи вимоги до діагностики та показників призначення. Ці вимоги стали основою для подальшої розробки, забезпечуючи чітке розуміння необхідних функціональних можливостей та критеріїв якості системи.

Далі, було визначено функціональні та нефункціональні вимоги, які дозволили деталізувати очікувану поведінку системи та її характеристики, такі як продуктивність, надійність та зручність використання. Розроблені сценарії використання системи наочно продемонстрували взаємодію користувача з додатком, відображаючи його ключові функції, включаючи запуск аналізу трафіку, вибір моделей та режимів роботи (реальний час/офлайн), збереження результатів, а також їх зупинку.

Ключова частина розділу присвячена розробці програмної частини. Було розглянуто етапи попередньої обробки набору даних, що є критично важливим для якості навчання моделей машинного навчання. Далі, детально описано реалізацію алгоритмів машинного навчання, включаючи використання таких передових архітектур, як рекурентні нейронні мережі (PHM), зокрема Long Short-Term Memory (LSTM), та ансамблеві методи, такі як Bagging, Boosting та Stacking, з фокусом на моделі CatBoost. Описана методологія та принцип роботи таких моделей як метод опорних векторів (SVM), що дозволяє ефективно класифікувати дані.

Важливим кроком була оцінка навчання моделі машинного навчання, яка підтвердила ефективність застосованих підходів. Проведене A/B тестування, як показано в Таблиці 2, чітко продемонструвало, що балансування даних суттєво покращує точність навчених моделей, що є критично важливим для виявлення рідкісних подій, таких як атаки.

Крім того, було розглянуто реалізацію аналізатора пакетів, що є основою для збору та обробки мережевого трафіку. Розробка користувацького інтерфейсу

забезпечила зручну та інтуїтивно зрозумілу взаємодію з системою.

Завершальним етапом було тестування додатка, яке підтвердило відповідність системи заявленим функціональним вимогам. Тестування, проведене за методологією TDD, охопило перевірку вибору моделей, коректність класифікації, ведення журналу, зміну мережевого інтерфейсу, збереження записів та можливість повторного запуску аналізу, що свідчить про високу стабільність та функціональність розробленого програмного застосунку.

ВИСНОВКИ

У даній кваліфікаційній роботі було успішно розроблено програмний застосунок для аналізу мережевого трафіку в режимі реального часу, використовуючи сучасні методи машинного навчання. Це досягнення стало можливим завдяки послідовному вирішенню низки важливих завдань. Спочатку було проведено ґрунтовний аналіз предметної області, що дозволило глибоко зрозуміти специфіку мережевого трафіку та різноманітність кібератак. На основі цього аналізу було підготовлено та попередньо оброблено набір даних, що має вирішальне значення для якості навчання моделей.

Наступним кроком був вибір та реалізація моделей машинного навчання. Були обрані та імплементовані такі архітектури, як рекурентні нейронні мережі (зокрема, LSTM), а також ансамблеві методи, включаючи Random Forest, AdaBoost, CatBoost та SVM. Детальний розгляд їхньої архітектури та принципів роботи підкреслив їхню ефективність у задачі виявлення аномалій. Проектування системи аналізу трафіку в реальному часі охопило розробку її архітектури та визначення взаємодії всіх компонентів, що забезпечило логічну та ефективну структуру застосунку.

Розроблена система була успішно реалізована, включаючи аналізатор пакетів та інтуїтивно зрозумілий користувацький інтерфейс. Цей інтерфейс дозволяє користувачеві обирати моделі машинного навчання, перемикатися між режимами роботи (реальний час та офлайн), а також зберігати результати аналізу. Завершальним етапом стала оцінка якості розробленого застосунку. Функціональне тестування підтвердило відповідність системи заявленим вимогам, а проведене A/B тестування моделей чітко продемонструвало значне підвищення точності класифікації після балансування даних, що є критично важливим для надійного виявлення рідкісних, але небезпечних подій, таких як мережеві атаки.

В цілому, розроблений застосунок демонструє високий потенціал для практичного застосування у сфері кібербезпеки, надаючи ефективний інструмент для моніторингу та виявлення мережевих загроз. Проте, існують подальші напрямки для його вдосконалення. Планується розширити список використовуваних методів

машинного навчання, збільшити функціональність, додати підтримку консольного режиму роботи та забезпечити загальний інтерфейс для отримання характеристик пакетів, що дозволить інтегрувати ще більше моделей. Також у найближчих планах – розробка та впровадження механізмів запобігання вторгненням. Для подальшого підвищення якості класифікації розглядається можливість створення глибших архітектур нейронних мереж, збільшення обсягу та різноманітності навчальних даних, а також більш ретельний аналіз та відбір використовуваних ознак трафіку. Реалізація цих перспективних планів дозволить створити ще більш потужний та адаптивний інструмент для забезпечення мережевої безпеки в умовах динамічного зростання кількості та складності кібератак..

ПЕРЕЛІК ПОСИЛАНЬ

1. DDoS attacks in Q1 2021. [Електронний ресурс] – Режим доступу до ресурсу: https://se-curelist.com/ddos-attacks-in-q1-2021/102166/.

2. DDoS attacks in Q3 2022. [Електронний ресурс] – Режим доступу до ресурсу: https://se-curelist.com/ddos-report-q3-2022/107860/.

3. DDoS Attack Trends and Insights in 2023 [Електронний ресурс] – Режим доступу до ресурсу: https://ddos-guard.net/en/blog/ddos-attack-trends-2023.

4. Meng X., Lin C., Wang Y., Zhang Y. Generative Pretrained Trans-former for Network Traffic // arXiv.org. 2023.

5. Abreu D., Abelem A. OMINACS: Online ML-based IOT network attack detection and classification system. // 2022 IEEE Latin-American Conference on Communications (LATINCOM), 2022. – С. 1–6.

6. Jonathan O., Misra S., Osamor V. Comparative analysis of machine learning techniques for network traffic classification. // IOP Conference Series: Earth and Environmental Science, 2021. – Т. 655. – № 1. – С. 12–25..

7. Kumari K., Mrunalini M. Detecting denial of service attacks using machine learning algorithms. // Journal of Big Data, 2022. – Т. 9. – № 1. – 56–73..

8. What is Transmission Control Protocol TCP/IP? [Електронний ресурс] – Режим доступу до ресурсу: https://www.fortinet.com/resources/cyberglossary/tcp-ip.

9. Cisco WAN and Application Optimization Solution Guide [Електронний ресурс] – Режим доступу до ресурсу: https://www.cisco.com/c/en/us/td/docs/nsite/enterprise/wan.

10. Understanding LSTM Networks [Електронний ресурс] – Режим доступу до ресурсу: https://colah.github.io/posts/2015-08-Understanding-LSTMs.

11. Intrusion detection evaluation dataset (CIC-IDS2017) [Електронний ресурс] – Режим доступу до ресурсу: https://www.unb.ca/cic/datasets/ids-2017.html.

12. CSE-CIC-IDS2018 on AWS. [Електронний ресурс] – Режим доступу до ресурсу: https://www.unb.ca/cic/datasets/ids-2018.html.

13. A Comprehensive Guide to Ensemble Learning (with Python codes) [Електронний ресурс] – Режим доступу до ресурсу: https://www.analyt-

icsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models.

14. Visual Studio Code – Code Editing [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com

15. PyQT6. [Електронний ресурс] – Режим доступу до ресурсу: https://www.riverbankcomputing.com/static/Docs/PyQt6

ДОДАТОК А

Текст програми нейромережевий аналіз структури інтернет-трафіку для класифікації мережевої активності

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

НЕЙРОМЕРЕЖЕВИЙ АНАЛІЗ СТРУКТУРИ ІНТЕРНЕТ-ТРАФІКУ ДЛЯ КЛАСИФІКАЦІЇ МЕРЕЖЕВОЇ АКТИВНОСТІ

Текст програми

804.02070743.25018-01 12 01

Листів 10

2025

АНОТАЦІЯ

Дана програма є десктопною комп'ютерною системою, розробленою для аналізу мережевого трафіку в режимі реального часу та офлайн. Її основне призначення — виявлення атак на основі методів машинного навчання, що сприяє підвищенню інформаційної безпеки.

Програма дозволяє користувачеві обирати між різними моделями машинного навчання, такими як Random Forest, LSTM, SVM, Decision Tree та Naive Bayes, для проведення аналізу. Вона підтримує два режими роботи: аналіз трафіку в реальному часі через мережеві інтерфейси (включаючи імітовані атакуючі інтерфейси для тестування) та офлайн-аналіз шляхом завантаження файлів мережевого трафіку (наприклад, .pcap файлів). Система надає детальну інформацію про хід аналізу в текстовому журналі, а також дозволяє зберігати ці логи для подальшого вивчення.

Розроблена з використанням Python та бібліотеки PyQt6 для графічного інтерфейсу, програма забезпечує кросплатформенність та інтуїтивно зрозумілий користувацький досвід. Вона може бути використана фахівцями з кібербезпеки, мережевими адміністраторами, дослідниками та студентами для моніторингу мереж, виявлення аномалій, тестування систем виявлення вторгнень та навчання в галузі аналізу мережевого трафіку.

ЗМІСТ

Main.py

```
import sys
from PyQt6.QtWidgets import (
    QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout,
    QPushButton, QComboBox, QCheckBox, QLineEdit, QTextEdit,
    QLabel, QFileDialog, QStackedLayout
)
from PyQt6.QtCore import Qt, QThread, pyqtSignal
from PyQt6.QtGui import QIcon  # Для іконок, якщо потрібно


# Приклад заглушки для Worker класу (реальний Worker клас буде складнішим)
class Worker(QThread):
    """
    Заглушка класу Worker для імітації аналізу трафіку.
    В реальному застосунку тут буде складна логіка взаємодії зі Scapy,
    обробка пакетів та класифікація за допомогою моделей ML.
    """
    # Сигнали для оновлення інтерфейсу
    log_message = pyqtSignal(str)
    analysis_complete = pyqtSignal()

    def __init__(self, selected_model, selected_mode, source_path):
        super().__init__()
        self.selected_model = selected_model
        self.selected_mode = selected_mode
        self.source_path = source_path
        self._is_running = True

    def run(self):
        self.log_message.emit(f"Запуск аналізу трафіку. Режим: {self.selected_mode},
Модель: {self.selected_model}")
        if self.selected_mode == "Live":
            self.log_message.emit(f"Прослуховування інтерфейсу: {self.source_path}...")
            # Імітація роботи AsyncSniffer
            for i in range(5):  # Імітуємо 5 секунд аналізу
                if not self._is_running:
                    break
                self.log_message.emit(f"Аналіз у реальному часі... Пакет {i + 1}")
                self.msleep(1000)  # Затримка на 1 секунду
            self.log_message.emit("Аналіз у реальному часі завершено.")
        elif self.selected_mode == "Offline":
            self.log_message.emit(f"Аналіз файлу: {self.source_path}...")
            # Імітація обробки файлу
```

```
            for i in range(3):  # Імітуємо 3 секунди аналізу
                if not self._is_running:
                    break
                self.log_message.emit(f"Аналіз офлайн... Порція даних {i + 1}")
                self.msleep(1500)  # Затримка на 1.5 секунди
            self.log_message.emit("Аналіз офлайн файлу завершено.")

        self.analysis_complete.emit()

    def stop(self):
        self._is_running = False
        self.log_message.emit("Запит на зупинку аналізу...")


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Застосунок для аналізу мережевого трафіку")
        self.setGeometry(100, 100, 800, 600)  # x, y, width, height

        self.worker_thread = None  # Для зберігання екземпляра Worker
        self.selected_model = None
        self.selected_work_mode = "Live"  # Початковий режим
        self.source_path = None  # Шлях до інтерфейсу або файлу

        self._init_ui()

    def _init_ui(self):
        central_widget = QWidget()
        self.setCentralWidget(central_widget)
        main_layout = QVBoxLayout(central_widget)

        # 1. Панель управління (Controls)
        controls_layout = QHBoxLayout()
        main_layout.addLayout(controls_layout)

        # Кнопка "Run Detection"
        self.run_button = QPushButton("Запустити виявлення")
        self.run_button.clicked.connect(self._run_button_was_clicked)
        controls_layout.addWidget(self.run_button)

        # Випадаючий список "model"
        controls_layout.addWidget(QLabel("Модель:"))
        self.model_combobox = QComboBox()
```

```
        self.model_combobox.addItems(
            ["Random Forest", "LSTM", "SVM", "Decision Tree", "Naive Bayes"])  # Приклад моделей
        self.model_combobox.currentIndexChanged.connect(self._update_selected_model)
        controls_layout.addWidget(self.model_combobox)

        # Кнопка "Stop"
        self.stop_button = QPushButton("Зупинити")
        self.stop_button.setEnabled(False)  # Початково неактивна
        self.stop_button.clicked.connect(self._stop_button_was_clicked)
        controls_layout.addWidget(self.stop_button)

        # Додатковий простір
        controls_layout.addStretch(1)

        # 2. Панель вибору режиму (Mode Selection)
        mode_selection_layout = QHBoxLayout()
        main_layout.addLayout(mode_selection_layout)

        # Чекбокс "Use offline analysis"
        self.work_mode_checkbox = QCheckBox("Використовувати офлайн-аналіз")
        self.work_mode_checkbox.stateChanged.connect(self._activate_check_box)
        mode_selection_layout.addWidget(self.work_mode_checkbox)

        # Група для вибору інтерфейсу/файлу (використовуємо QStackedLayout для перемикання)
        self.work_mode_layout = QStackedLayout()
        mode_selection_layout.addLayout(self.work_mode_layout)

        # 2.1. Режим "Live" (інтерфейс)
        live_mode_widget = QWidget()
        live_mode_layout = QHBoxLayout(live_mode_widget)
        live_mode_layout.setContentsMargins(0, 0, 0, 0)  # Видалити зайві відступи
        live_mode_layout.addWidget(QLabel("Інтерфейс:"))
        self.interface_combobox = QComboBox()
        self.interface_combobox.addItems(["eth0", "wlan0", "lo", "any"])  # Приклад інтерфейсів
        self.interface_combobox.currentIndexChanged.connect(self._update_source_path)
        live_mode_layout.addWidget(self.interface_combobox)
        self.work_mode_layout.addWidget(live_mode_widget)

        # 2.2. Режим "Offline" (вибір файлу)
        offline_mode_widget = QWidget()
        offline_mode_layout = QHBoxLayout(offline_mode_widget)
```

```
        offline_mode_layout.setContentsMargins(0, 0, 0, 0)
        self_file_label = QLabel("Шлях до файлу:")
        self.file_path_edit = QLineEdit()
        self.file_path_edit.setReadOnly(True)  # Шлях обирається через кнопку
        self.file_path_edit.setText("Не обрано")
        self.select_file_button = QPushButton("Обрати файл")
        self.select_file_button.clicked.connect(self._select_file)
        offline_mode_layout.addWidget(self_file_label)
        offline_mode_layout.addWidget(self.file_path_edit)
        offline_mode_layout.addWidget(self.select_file_button)
        self.work_mode_layout.addWidget(offline_mode_widget)

        # Встановлюємо початковий активний режим (Live)
        self.work_mode_layout.setCurrentIndex(0)

        # Переміщено сюди: 3. Текстове поле для логів
        self.log_text_edit = QTextEdit()
        self.log_text_edit.setReadOnly(True)
        main_layout.addWidget(QLabel("Журнал аналізу та системні повідомлення:"))
        main_layout.addWidget(self.log_text_edit)

        # Переміщено сюди: 4. Кнопка "Save Logs"
        self.save_logs_button = QPushButton("Зберегти логи")
        self.save_logs_button.clicked.connect(self._save_logs)
        main_layout.addWidget(self.save_logs_button)

        # Тепер можна безпечно викликати ці методи, оскільки log_text_edit вже існує
        self._update_selected_model()  # Ініціалізуємо обрану модель
        self._update_source_path()  # Ініціалізуємо шлях/інтерфейс

    def _update_selected_model(self):
        """Оновлює обрану модель при зміні в комбобоксі."""
        self.selected_model = self.model_combobox.currentText()
        self.log_text_edit.append(f"Обрано модель: {self.selected_model}")

    def _update_source_path(self):
        """Оновлює шлях до джерела даних (інтерфейс/файл)"""
        if self.selected_work_mode == "Live":
            self.source_path = self.interface_combobox.currentText()
            self.log_text_edit.append(f"Обрано інтерфейс для прослуховування:
{self.source_path}")
        # Для офлайн-режиму шлях оновлюється в _select_file

    def _activate_check_box(self, state):
```

```
        """Обробляє зміну стану чекбокса "Use offline analysis"."""
        if state == Qt.CheckState.Unchecked.value:  # 0
            self.work_mode_layout.setCurrentIndex(0)  # Переключаємо на інтерфейс (Live)
            self.selected_work_mode = "Live"
            self.log_text_edit.append("Режим аналізу: Реальний час")
            self._update_source_path()  # Оновити шлях відповідно до нового режиму
        elif state == Qt.CheckState.Checked.value:  # 2
            self.work_mode_layout.setCurrentIndex(1)  # Переключаємо на вибір файлу (Offline)
            self.selected_work_mode = "Offline"
            self.log_text_edit.append("Режим аналізу: Офлайн")
            # Шлях до файлу буде встановлено, коли користувач його обере
            self.source_path = self.file_path_edit.text()  # Беремо поточний текст, якщо він є

        # Скидаємо шлях до файлу, якщо переключились на Live-режим
        if self.selected_work_mode == "Live":
            self.file_path_edit.setText("Не обрано")
            self.source_path = self.interface_combobox.currentText()  # Переконаємось, що джерело оновлено

    def _select_file(self):
        """Викликає діалогове вікно для вибору файлу."""
        file_name, _ = QFileDialog.getOpenFileName(self, "Вибрати файл для аналізу", "",
                                                   "PCAP Files (*.pcap *.pcapng);;All Files (*)")
        if file_name:
            self.file_path_edit.setText(file_name)
            self.source_path = file_name
            self.log_text_edit.append(f"Обрано файл для аналізу: {self.source_path}")
        else:
            self.log_text_edit.append("Вибір файлу скасовано.")
            self.file_path_edit.setText("Не обрано")
            self.source_path = None

    def _run_button_was_clicked(self):
        """Обробляє натискання кнопки "Запустити виявлення"."""
        if self.worker_thread and self.worker_thread.isRunning():
            self.log_text_edit.append("Аналіз вже запущено. Будь ласка, зупиніть поточний процес.")
            return

        if self.selected_work_mode == "Offline" and not self.source_path or self.file_path_edit.text() == "Не обрано":
            self.log_text_edit.append("Помилка: Будь ласка, оберіть файл для офлайн-аналізу.")
```

```
            return

        self.log_text_edit.append("---------- Запуск нового аналізу ----------")

        # Блокуємо кнопки та запускаємо потік
        self.run_button.setEnabled(False)
        self.stop_button.setEnabled(True)
        self.model_combobox.setEnabled(False)
        self.work_mode_checkbox.setEnabled(False)
        self.interface_combobox.setEnabled(False)
        self.select_file_button.setEnabled(False)

        # Створення та запуск потоку Worker
        self.worker_thread = Worker(self.selected_model, self.selected_work_mode,
self.source_path)
        self.worker_thread.log_message.connect(self._append_log_message)
        self.worker_thread.analysis_complete.connect(self._analysis_thread_complete)
        self.worker_thread.start()

    def _stop_button_was_clicked(self):
        """Обробляє натискання кнопки "Зупинити"."""
        if self.worker_thread and self.worker_thread.isRunning():
            self.worker_thread.stop()
            self.stop_button.setEnabled(False)  # Можна деактивувати кнопку зупинки
одразу
        else:
            self.log_text_edit.append("Аналіз не запущено.")

    def _analysis_thread_complete(self):
        """Викликається, коли потік аналізу завершує роботу."""
        self.log_text_edit.append("---------- Аналіз завершено ----------")
        self.run_button.setEnabled(True)
        self.stop_button.setEnabled(False)
        self.model_combobox.setEnabled(True)
        self.work_mode_checkbox.setEnabled(True)

        # Активуємо відповідні елементи залежно від режиму
        if self.selected_work_mode == "Live":
            self.interface_combobox.setEnabled(True)
        else:  # Offline
            self.select_file_button.setEnabled(True)

    def _append_log_message(self, message):
        """Додає повідомлення до текстового поля логів."""
```