Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА

## Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи машинного навчання на основі аналізу трафіка для виявлення програм-вимагачів платформи Android
(тема)


Виконав:

студент II курсу, групи КСМм-23-1

Ступаренко Р. Ю.
(прізвище, ініціали)


Спеціальність

123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Комп'ютерні системи та мережі
(повна назва освітньої програми)

Керівник: ас. Кравченко П.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерні системи та мережі
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

# ЗАВДАННЯ

## НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Ступаренку Роману Юрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Методи машинного навчання на основі аналізу трафіка для виявлення програм-вимагачів платформи Android

затверджена наказом по університету від “ 22 ” листопада 2024 р. № 1237 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 15 січня 2025 р.

3. Вхідні дані до роботи
Штучний інтелект, машинне навчання, Android, програми-вимагачі, аналіз трафіка.

4. Перелік питань, що потрібно опрацювати у роботі
1. Аналіз предметної області
2. Методологія досліджень
3. Експерементальні дослідження
Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Слайд-презентація – 15 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1 )

| Найменування розділу | Консультант (посада, прізвище, ім'я, по батькові) | Позначка консультанта про виконання розділу | |
|---|---|---|---|
| | | підпис | дата |
| | | | |
| | | | |

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання | 26.11.24 | |
| 2 | Аналіз літератури | 26.11.24-01.12.24 | |
| 3 | Дослідження обраних методів МН | 02.12.24-12.12.24 | |
| | Програмна реалізація обраних методів | 13.12.24-20.12.24 | |
| | Аналіз результатів експериментів | 21.12.24-23.12.24 | |
| | Оформлення пояснювальної записки | 24.12.24-29.12.24 | |
| | Проходження перевірки на плагіат | 30.12.24-02.01.25 | |
| | Отримання рецензії | 03.01.25-13.01.25 | |
| | Подача роботи в ЕК | 14.01.25-15.01.25 | |
| | | | |
| | | | |
| | | | |

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ ас. Кравченко П.О.
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 67 с., 6 рис., 4 табл., 2 дод., 49 джерел.

ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, ANDROID, ПРОГРАМИ-ВИМАГАЧІ, АНАЛІЗ ТРАФІКА.

Метою кваліфікаційної роботи є використання методів машиного навчання на базі аналізу трафіка для виявлення програм-вимагачів на платформі Android.

У ході виконання кваліфікаційної роботи було надано комплексний огляд літератури, щоб продемонструвати існуючі дослідження з метою аналізу прогалин і пошуку нових напрямів дослідження. Для проведення двох експериментів було використано останній набір даних виявлення програм-вимагачів Android, 2023 від Kaggle. У рамках попередньої обробки даних для усунення дисбалансу набору даних була прийнята техніка рандомізованої недостатньої вибірки. Після попередньої обробки даних було застосовано вибір ознак за допомогою прямого вибору ознак і важливості ознак. Всього 19 функцій були визнані вирішальними для аналізу та ідентифікації атак. Було проведено два експерименти з використанням DT, SVM, KNN, моделі ансамблю (DT, SVM, KNN), FNN і TabNet.

# ABSTRACT

Master’s thesis: 67 pages, 6 figures, 4 tables, 2 appendices, 49 sources.

ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, ANDROID, RANSOMWARE, TRAFFIC ANALYSIS.

The major goal of this thesis is to use machine learning methods based on traffic analysis to detect ransomware on the Android platform.

In order to qualification work, a comprehensive literature review was provided to demonstrate existing research in order to analyze gaps and find new research directions. The latest Android ransomware detection dataset, 2023 from Kaggle, was used to conduct two experiments. As part of the data preprocessing, a randomized undersampling technique was adopted to eliminate the imbalance of the dataset. After the data preprocessing, feature selection was applied using direct feature selection and feature importance. A total of 19 features were found to be crucial for analyzing and identifying attacks. Two experiments were conducted using DT, SVM, KNN, ensemble model (DT, SVM, KNN), FNN, and TabNet.

ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – штучний інтелект

МН – машинне навчання

ANDROID – операційна система для мобільних пристроїв

DT – дерева рішень

KNN – k-найближчих сусідів

SVM – метод опорних векторів

# ВСТУП

У сучасну цифрову епоху використання пристроїв Android широко поширене в різних секторах. Кіберзлочинці неминуче пристосовуються до нових технологій безпеки та використовують ці платформи для використання вразливостей у підлих цілях, таких як викрадення конфіденційних і особистих даних користувачів. Це може призвести до фінансових втрат, дискредитації, програм-вимагачів або розповсюдження інфекційного шкідливого програмного забезпечення та інших катастрофічних кібератак. Через те, що програми-вимагачі шифрують дані користувача та вимагають викуп в обмін на ключ дешифрування, це один із найруйнівніших типів шкідливого програмного забезпечення. Наслідки атак програм-вимагачів можуть варіюватися від втрати важливих даних до зриву бізнес-операцій і значної грошової шкоди. Методи на основі штучного інтелекту (ШІ), а саме машинне навчання (ML), довели свою ефективність у виявленні атак програм-вимагачів Android. Проте ансамблевы моделі та моделі глибокого навчання (DL) не були достатньо вивчені. Тому в цій кваліфікаційній роботі використовувуються методи на основі ML і DL для створення ефективних, точних і надійних моделей для двійкової класифікації. Для навчання та тестування моделей використовувався загальнодоступний набір даних від Kaggle, що складається з 392035 записів із безпечним трафіком і 10 різними типами атак програм-вимагачів Android.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Основні принципи функціонування Android та питання безпеки

Швидке зростання використання мобільних пристроїв зробив Android однією з провідних операційних систем для смартфонів і планшетів. Станом на липень 2024 року Android має найвищу частку ринку – 70,8% [1]. Android – це мобільна операційна система з відкритим кодом на базі Linux, розроблена Google [2]. Android 14 – остання версія, випущена в 2023 році. Вона підтримує багато технологій, як-от Wi-Fi, службу коротких повідомлень (SMS), Bluetooth, акселерометри, камеру, глобальні системи позиціонування (GPS), передачу голосу через LTE (VoLTE) тощо. Завдяки своїй відкритості Android став надзвичайно популярним серед розробників і споживачів. Крім того, розробники програмного забезпечення можуть швидко змінити та оновити його відповідно до найновіших стандартів мобільних технологій. На жаль, ця популярність також приваблює кіберзлочинців, які використовують приватні дані та персональну інформацію користувачів без їхньої згоди [3]. Однією з найпоширеніших і руйнівних атак, спрямованих на пристрої Android, є програми-вимагачі. Програми-вимагачі – це різновид шкідливих програм, які шифрують файли на пристрої та вимагають плату за їх розшифровку. Як правило, кіберзлочинці вимагають оплату в криптовалютах, таких як біткойни та інші монети, щоб уникнути виявлення. Типовий сценарій атак програм-вимагачів часто починається, коли користувач завантажує шахрайську програму з магазину Google Play або альтернативного стороннього ринку [4]. Зі збільшенням кількості пристроїв Android і відкритим характером платформи, яка дозволяє легко завантажувати програми з неофіційних джерел, серйозність атак програм-вимагачів досягла загрозливого рівня. У такій ситуації відновлення даних може бути складним завданням, і підвищується

ризик подальших атак і крадіжки особистих даних, що призводить до зниження довіри користувачів до рішень безпеки [5, 6]. Атаки програм-вимагачів зазвичай спрямовані на певні галузі. У 2023 році виробничі компанії в усьому світі зазнали 628 таких атак, а сектор харчових продуктів і напоїв уважно слідкував за ними з понад 50 інцидентами програм-вимагачів. Що стосується поширення атак програм-вимагачів на критичну інфраструктуру, Північна Америка посіла лідерство серед глобальних регіонів, а Європа посідає друге місце [7]. Крім того, глобальна кількість атак програм-вимагачів за рік з 2018 по 2023 рік така: у 2023 році організації виявили приголомшливі 513,34 мільйона атак програм-вимагачів у всьому світі.

Переходимо до того, як програми-вимагачі розповсюджуються на пристроях Android, важливо знати, що програми-вимагачі постійно розширюються завдяки вдосконаленим можливостям шифрування, витісняючи усталені стандарти, такі як фішинг, банківські трояни, розподілена відмова в обслуговуванні (DDoS) і крипто-злом. Однак злочинці використовують ці моделі як початкову сходинку, потім посилюють атаку та зрештою здійснюють цілеспрямовані атаки, таким чином змушуючи жертви платити. Відкривши вкладення електронної пошти або натиснувши оголошення, перейшовши за посиланням або навіть перейшовши на веб-сайт із вбудованим зловмисним програмним забезпеченням, можна несвідомо завантажити програму-вимагач на електронний пристрій. Коли код завантажується на пристрій, він блокує доступ до пристрою та будь-яких збережених файлів і даних. Більш деструктивні версії можуть шифрувати дані на мережевих пристроях, а також на локальних і підключених дисках. Користувачі виявляють це, коли їхні дані стають недоступними або коли з’являються повідомлення з інформацією про атаку та вимогою викупу. Злочинці погрожують оприлюднити конфіденційні дані, якщо їхні жертви не сплатять протягом зазначеного часу або вирішать відновити зашифровані дані за допомогою резервних копій. Деякі зловмисники навіть продають

конфіденційні дані на аукціоні в темній мережі. Важливо знати, що атаки програм-вимагачів мають багато різних проявів і проявляються в усіх формах і розмірах. Є дві основні категорії, а саме блокування екрана та крипто [8]. На екрані блокування програма-вимагач блокує доступ до системи, стверджуючи, що система зашифрована. Програми-вимагачі для екрану блокування зазвичай не націлені на критичні файли; зазвичай він хоче лише заблокувати користувача. З іншого боку, крипто-вимагач шифрує дані в системі, такі як документи, зображення та відео, роблячи вміст марним без ключа дешифрування та без втручання в основні функції пристрою. Користувачі можуть бачити свої файли, але не можуть отримати до них доступ, якщо не сплатять вимогу викупу, інакше всі їхні файли будуть видалені. Деякі приклади програм-вимагачів Android включають Android/Simplocker, Android/Lockerpin, WannaLocker тощо [8]. Отже, існує гостра потреба в розробці ефективних методів виявлення програм-вимагачів Android, щоб протистояти цій зростаючій небезпеці.

Пристрої Android оснащені різноманітними вбудованими датчиками, які вимірюють рух, орієнтацію та інші фактори навколишнього середовища. Вбудованими датчиками керує платформа сенсорів Android. Датчики бувають апаратними або програмними. Апаратні датчики – це фізичні компоненти, інтегровані в планшет або телефон. Вони отримують свої дані, безпосередньо сприймаючи певні сигнали. Програмні датчики не є фізичними пристроями. Програмні датчики, які також називаються віртуальними або синтетичними датчиками, отримують дані від одного або кількох апаратних датчиків. Операційна система Android підтримує три основні категорії датчиків (датчики руху, навколишнього середовища, положення). Датчики руху вимірюють силу обертання та силу прискорення в трьох вимірах. Датчики вектора обертання, гіроскопи, акселерометри та датчики гравітації входять до цієї категорії. Датчики навколишнього середовища контролюють низку факторів навколишнього середовища, таких як вологість, освітленість, тиск навколишнього повітря та температура.

Термометри, фотометри та барометри підпадають під цю категорію. Датчики положення вимірюють фізичне положення пристрою. До цієї категорії належать магнітометри та датчики орієнтації. Датчики Android надають дані як серію подій датчиків. Ці вбудовані датчики широко використовуються в багатьох функціях програм сторонніх розробників. Однак програми сторонніх розробників можуть зчитувати дані з вбудованих датчиків, не вимагаючи жодних дозволів. Це може призвести до проблем із безпекою. Конфіденційність користувачів може бути порушена добре розробленими шкідливими програмами, які використовують вбудовані датчики. Зламаний пристрій через успішну атаку програми-вимагача потенційно може мати ширші наслідки для безпеки. Якщо програмне забезпечення-вимагач є частиною більш складної атаки, воно може вимагати додаткових дозволів або використовувати вразливості в операційній системі пристрою. Це може опосередковано вплинути на різні функції, зокрема датчики. Зловмисник може спробувати маніпулювати датчиками для несанкціонованого доступу або спостереження. Успішно виявивши атаки програм-вимагачів, можна зменшити площу атаки. Крім того, це обмежить несанкціонований доступ до датчиків Android.

Існують різні типи аналізу функцій для програм-вимагачів. Статичний аналіз залежить від функцій, зібраних без виконання будь-якого коду, тоді як динамічний аналіз отримує функції на основі виконання коду (або емуляції). Статичний аналіз програми Android може залежати від функцій, отриманих із файлу маніфесту або байт-коду Java. Оскільки виконання коду не потрібне, статичний аналіз часто вважається більш ефективним. Тим часом такі функції, як динамічне завантаження коду та системні виклики, які збираються під час роботи програми, можна вирішити за допомогою динамічного аналізу програми Android. Динамічний аналіз більш інформативний, оскільки оцінюється виконуваний код [9]. Аналіз мережевого трафіку в контексті динамічного аналізу зосереджується на моніторингу та аналізі мережевого трафіку, створеного пристроєм Android,

для виявлення будь-яких підозрілих моделей і зловмисної поведінки, пов’язаної з діяльністю програм-вимагачів. Програми-вимагачі часто використовують мережеві з’єднання для передачі ключів шифрування, зв’язку з командно-контрольними серверами та допомоги у викраденні даних або процедурі оплати. Відстежуючи дані, якими обмінюються пристрій і зовнішні об’єкти, аналіз мережевого трафіку прагне виявити ці шахрайські дії. Нарешті, гібридний аналіз – це стратегія аналізу, яка поєднує статичний і динамічний аналізи для компенсації їхніх окремих недоліків [10].

Починаючи з попереднього десятиліття, у дисципліні кібербезпеки було значно використано методи ШІ, а саме ML і DL [11-16]. Потенціал цих методів полягає в тому, щоб вчитися на наданих даних і, як наслідок, отримувати цінну інформацію та правильно прогнозувати випадки в майбутньому. Забезпечуючи автоматичний та інтелектуальний аналіз складних шаблонів, функцій і поведінки в даних, ML і DL відіграють значну роль у виявленні шкідливих програм [17-19]. Ці підходи дозволяють системам безпеки точно виявляти та класифікувати зловмисне програмне забезпечення з високою точністю, навіть якщо зловмисне програмне забезпечення розвивається та стає все складнішим. Ефективним рішенням для захисту користувачів від конкретних атак програм-вимагачів є методи на основі машинного навчання [20]. ML можна використовувати для виявлення аномальних дій, пов’язаних з атаками програм-вимагачів. Рішення безпеки на основі ML можна використовувати для аналізу подій у мережі та сповіщення адміністраторів системи про можливі загрози атак. Крім того, алгоритми ML можуть вивчати та аналізувати дані про минулі атаки програм-вимагачів і створювати моделі, здатні краще передбачати подібні майбутні атаки, що зрештою допомагає підвищити безпеку систем. Хоча програмне забезпечення-вимагач є постійною глобальною загрозою, воно продовжує розвиватися, через що традиційні методи виявлення, як-от підходи на основі сигнатур, є недостатніми. У результаті дослідники та експерти з безпеки звернулися до ML як багатообіцяючої альтернативи для виявлення та

пом’якшення атак програм-вимагачів. Методи машинного навчання набули популярності у сфері виявлення та класифікації програм-вимагачів Android [21]. Ці підходи використовують функції, такі як шаблони мережевого трафіку, системні виклики та ентропію файлів, щоб відрізнити нормальну поведінку від зловмисної діяльності [22]. Методи DL, зокрема, продемонстрували надзвичайну ефективність у виявленні різних форм зловмисного програмного забезпечення, включаючи програми-вимагачі, у боротьбі з динамічною природою цих загроз.

У той час як у попередніх дослідженнях вивчалися методики на основі машинного навчання для виявлення програм-вимагачів Android, все ще існує потреба у всебічному дослідженні їх ефективності та точності. Оскільки атаки програм-вимагачів на пристрої Android продовжують зростати, модель на основі ML має потенціал для ефективного виявлення та запобігання таким атакам. Наскільки ефективна модель на основі ML у виявленні програм-вимагачів Android. Це питання дослідження має вирішальне значення, оскільки атаки програм-вимагачів на пристрої Android стають дедалі поширенішими, і ефективна модель на основі ML потенційно може допомогти у виявленні та запобіганні таким атакам. Таким чином, це дослідження спрямоване на оцінку ефективності методів ML і DL у виявленні програм-вимагачів Android.

Це дослідження спеціально зосереджено на методах: дерева рішень (DT), опорних векторів (SVM), k-найближчих сусідів (KNN), сукупності цих трьох алгоритмів (DT, SVM, KNN), FNN і TabNet, використовуючи останній набір даних, доступний на Kaggle. KNN — це простий алгоритм ML на основі екземплярів, який застосовується для цілей класифікації та регресії. Він класифікує точки даних на основі більшості класів їхніх K-найближчих сусідів у просторі ознак [23]. Навпаки, DT – це контрольований алгоритм навчання, який створює деревоподібну структуру для прийняття рішень [24]. SVM, з іншого боку, є надійним алгоритмом класифікації та регресії, який визначає гіперплощину, яка оптимально розділяє класи у високовимірному

просторі та може вмістити нелінійні дані через функції ядра [25]. FNN, також відомий як багатошаровий персептрон (MLP), являє собою штучну нейронну мережу (ШНМ) з односпрямованим потоком інформації, що охоплює кілька шарів взаємопов'язаних нейронів, і виконує різні завдання ML [26]. Нарешті, TabNet – це спеціалізована модель глибокої нейронної мережі, розроблена для табличних даних, що охоплює структуровану інформацію, яку зазвичай можна знайти в електронних таблицях. Він працює надзвичайно добре у захопленні складних взаємозв'язків у даних шляхом поєднання дерев рішень і механізмів уваги [27]. У відповідності з нашим комплексним підходом ми використали модель ансамблю, поєднуючи моделі DT, SVM і KNN. Ця стратегічна інтеграція дозволяє нам використовувати їхні колективні можливості прийняття рішень, підвищуючи здатність моделі охоплювати різноманітні аспекти даних і отримувати надійні результати.

Ключові завдання цієї роботи наступні:

- використання реального набору даних, що містить раніше невикористані зразки відомих варіантів програм-вимагачів. Цей автентичний набір даних підвищує надійність оцінювання, оскільки він представляє реальні сценарії, з якими стикаються користувачі, роблячи результати більш застосовними та значущими;

- визначення найважливіших функцій трафіку, які сприяють виявленню програм-вимагачів Android;

- представляємо модель ансамблю, яка поєднує в собі сильні сторони кількох алгоритмів ML. Цей комплексний підхід має на меті підвищити загальну точність виявлення та надійність проти нових загроз програм-вимагачів шляхом використання різноманітних точок зору та стратегій прийняття рішень кожної окремої моделі;

- дослідження та оцінка ефективності двох моделей DL, які не були розгорнуті в існуючій літературі в області виявлення програм-вимагачів Android.

## 1.2 Огляд літератури

У цьому розділі представлені дослідження в існуючій літературі за період 2018-2023 рр., які використовували методи ML і DL у сфері виявлення програм-вимагачів Android.

У більшості досліджень застосовувалися різні методи МЛ. В [28] запропонував статичний аналіз на рівні байтів для виявлення атак програм-вимагачів за допомогою моделі випадкового лісу (RF). Набір даних, що складається з 1680 виконуваних файлів (840: програми-вимагачі, 840: безпечні), використовувався для навчання та оцінки їх підходу. Набір даних було розділено на набори для навчання та тестування 50:50 з однаковою кількістю файлів програм-вимагачів і хороших програм, щоб уникнути дисбалансу. Запропонована система використовувала метод вибору ознак коефіцієнта підсилення, щоб знайти оптимальний розмір розміру між 1000 і 7000 характеристиками для побудови радіочастотної моделі. Було виявлено, що побудова РЧ-моделі з 1000 ознаками та 100 деревами дало найкращі результати з точки зору складності та точності часу, з часом прогнозування 1,37 с та точністю 97,74%. Велику кількість дерев можна вважати обмеженням для цього дослідження, враховуючи його підвищену складність.

В [29] запропонувано підхід на основі машинного навчання для виявлення та класифікації атак програм-вимагачів, використовуючи набір даних із 54 функціями та 138 047 зразками (96632: програми-вимагачі, 41415: доброякісні). Зі зразків 70% використаного набору даних були програмами-вимагачами, а решта 30% були законними, що демонструє явну проблему дисбалансу. Проте автори зосередилися на методах відбору ознак, включаючи поріг дисперсії та інфляцію дисперсії, щоб усунути ознаки з низьким варіантом і високою корельяцією. Зрештою, було вибрано 12 ознак для вивчення п’яти різних алгоритмів, а саме DT, RF, наївної байєсівської (NB), логістичної регресії (LR) і ANN для класифікації. Результати показали, що радіочастотний класифікатор перевершив інші моделі, досягнувши

найвищої точності 99%, запам’ятовування 97%, F-бета 97% і точності 99%.

В [30] і [31] використано набір даних HelDroid, щоб відрізнити програми-вимагачі Android від інших шкідливих програм. У роботі Вікторіано набір даних спочатку містив 1923 додатки для Android, а вилучення функцій призвело до 11 функцій. Використовуючи класифікатори DT, RF, посилення градієнта (GB) і Adaboost, класифікатор DT вирізнявся найвищою точністю 98,8% і низьким рівнем помилкових позитивних результатів (FPR) 0,2%. Спираючись на цей набір даних, запропонувано гібридну систему для виявлення програм-вимагачів, класифікуючи 50 функцій за допомогою алгоритмів J48, NB і LR. Гібридна система досягла точності 100% і FPR менше 4%, що підкреслює важливість поєднання статичного та динамічного аналізів. Обидва дослідження підкреслюють потребу в більш обширних і різноманітних наборах даних для підтвердження ефективності їхніх підходів щодо охоплення всіх програм-вимагачів і законних програм.

В [32] роботі надано комплексне рішення для боротьби зі зростаючою загрозою програм-вимагачів Android. Набір даних, зібраний з різних джерел, складався з 1000 програм (500: програми-вимагачі, 500: безпечні). Автори використали 116 дозволів як функцію, отриману за допомогою статичного аналізу. Для класифікації використовувалися чотири моделі ML, а саме RF, J48, NB і послідовна мінімальна оптимізація (SMO). RF перевершив, з точністю 96,9%. Запропонована модель могла стати цінним доповненням до існуючих систем виявлення програм-вимагачів. Однак обмеженням, яке слід розглянути, є ефективність цієї моделі, яка може бути знижена, якщо розробники програм-вимагачів знайдуть способи обійти запитувані дозволи або якщо з’являться нові типи програм-вимагачів, які вимагають інших методів виявлення. Потрібні подальші дослідження, щоб усунути ці потенційні обмеження та покращити загальну безпеку пристроїв Android.

В [33] представлено новий комп’ютеризований легкий підхід, відомий як RanDroid, для виявлення та пом’якшення загроз програм-вимагачів на

пристроях Android на основі статичного та динамічного аналізів. RanDroid витягував зображення та текст, щоб визначити наявні екрани блокування або погрозливі нотатки. Автори використовували набір даних із 1450 програм (950: програми-вимагачі, 500: безпечні). Для класифікації використовувався NB, що досягло точності 91%. Це дослідження дало багатообіцяючі результати щодо використання NB для виявлення програм-вимагачів, але потрібна подальша робота, щоб підвищити його точність і розширити його можливості. Крім того, використання передових методів, таких як обробка природної мови (NLP) і автоматичне виправлення орфографії, може покращити продуктивність моделі.

Крім того, в [34] представлено підхід виявлення програм-вимагачів Android із використанням динамічного аналізу. Автори поєднали загальнодоступний набір даних, а саме VirusTotal, і доброякісні програми з Google Play. Зрештою він складався з 800 програм (400: програми-вимагачі, 400: безпечні). Загалом 52 системні виклики були отримані як функції за допомогою динамічного аналізу. Для класифікації було використано три моделі ML, а саме RF, J48 і NB. Радіочастотність перевершила ефективність із істинним позитивним показником (TPR) 98%, FPR 1,6% і точністю 98,31%.

В [35] автори запропонували підхід до виявлення програм-вимагачів для Android версії 11, рівень API 30. Автори використовували сканери для безпечних програм із Google Play, а програми-вимагачі збирали з загальнодоступного набору даних RansImDS-API & Permissions. Загалом 302 дозволи та виклики пакетів API розглядалися як функції та використовувалися для аналізу 1000 програм (500: програми-вимагачі, 500: безпечні). Крім того, GR було застосовано для вибору 225 найкращих функцій. Використані класифікатори ML були RF, DT, SMO та NB. Для порівняння, РЧ отримав найвищу ефективність із точністю 98,2%, запам’ятовуванням 99% і точністю 97,4%. Виявилося, що з точки зору розміру DT є найменш складною моделлю.

Однак в [36] запропонував гібридний підхід для виявлення та знищення програм-вимагачів Android. Щоб отримати набір домінуючих ознак, у цьому дослідженні використовувався абсолютно новий алгоритм вибору домінантних ознак, який зрештою вибрав 60 найбільш домінуючих і прогнозованих ознак у наборі. Вони зібрали 3249 зразків програм (1763: програми-вимагачі, 1486: нешкідливі). Під час свого експерименту вони використовували кілька класифікаторів, таких як J48, дерево логістичної моделі (LMT), RF і випадкове дерево (RT). Результати дослідження показали, що їхній гібридний підхід мав рівень точності 99,85% для виявлення програм-вимагачів Android. Виявлено, що використання запропонованого методу відбору домінантних ознак для вибору найбільш релевантних ознак значно покращує класифікацію. Однак у цьому дослідженні також було кілька обмежень, зокрема невеликий розмір вибірки та відсутність різноманітності досліджуваних сімейств програм-вимагачів.

Цікаво, що масштабована структура, відома як система виявлення програм-вимагачів на основі інтерфейсу прикладного програмування (API-RDS), була розроблена в [4] для виявлення програм-вимагачів. Для проектів HelDroid і RansomProper, Virus Total, Koodous і Google Play було використано набір даних із 1000 програм (500: програми-вимагачі, 500: доброякісні). За допомогою платформи Weka побудована прогностична модель API-RDS. Вектор функцій складався зі 174 функцій викликів пакетів API, що належать Android API 27. Отже, дані були розділені за допомогою техніки 10-кратної перехресної перевірки. Класифікатори RF, SMO, J48 і NB були розгорнуті в першому експерименті. RF перевершує показники з точністю 97%, площа під кривою (AUC) 99,5% і каппа 94%. Крім того, у другому експерименті платформа успішно виявила 96% невидимих зразків програм-вимагачів і 97% невидимих доброякісних зразків із загальною точністю 96,5%. Було виявлено, що java.lang із середнім показником 15928,96 викликів був найбільш часто запитуваним і використовуваним пакетом у програмах-вимагачах.

Крім того, модель була представлена в [37] для системи виявлення вторгнень програм-вимагачів Android. Набір даних, що складається з 10 854 зразків (4354: програми-вимагачі, 6500: доброякісні), було використано з CICandMal2017. Було розглянуто та введено в моделі десять сімей програм-вимагачів, що складаються з 80 функцій. Було розгорнуто три класифікатори J48, NB і OneR. J48 показав найкращі результати з точністю 75,76%, запам'ятовуванням 75,73% і F-показником 75,7%. Крім того, використання класифікатора J48 забезпечило зменшений час виконання з меншою кількістю атрибутів. Було виявлено, що перші 10 атрибутів у кожному з десяти сімейств програм-вимагачів містять байти Init Win у зворотному напрямку та Init Win у байтах вперед.

В роботах [38,39] автори, використовуючи набір даних RansomProber для своїх досліджень виявлення програм-вимагачів Android. У своїй початковій роботі фреймворк RansomDroid використовував неконтрольоване машинне навчання на 2300 наборах пакетів Android (APK), досягнувши 98,08% точності за допомогою моделі суміші Гауса (GMM). Розширивши свої дослідження для виявлення локерів і програм-вимагачів, вони досягли 99,59% точності на наборі даних із 4076 програм, які використовують модель LR. У наступному дослідженні автори розробили структуру для класифікації доброякісних і шкідливих програм Android. Використовуючи набір даних із 4721 програми, RF-модель ансамблю досягла рівня точності 99,67% у завданні двійкової класифікації, демонструючи свій потенціал для ідентифікації програм-вимагачів у реальному часі на телефонах на базі Android. Набір функцій складався з 1045 функцій у цих дослідженнях.

Автори в [40] запропонував новий підхід до ідентифікації програм-вимагачів, який базується на техніці ML, заснованій на еволюції. Техніка передискретизації синтетичної меншості запропонованого методу (SMOTE-tBPSO-SVM) використовувала BPSO як алгоритм оптимізації та пошуку та лінійний SVM для класифікації та виявлення. Вони зібрали 10153 зразки програм (500: програми-вимагачі, 9653: безпечні) з різних джерел даних.

Використовуючи 182 дозволи та виклики API як функції, продуктивність запропонованого підходу SMOTE-tBPSO-SVM перевершила звичайні алгоритми машинного навчання з точки зору чутливості (96,4%), специфічності (99,9%) і g-середнього значення (97,5%).

Щоб усунути обмеження, пов’язані з низькою точністю виявлення та високою обчислювальною складністю в існуючих рішеннях для виявлення програм-вимагачів Android, в [41] запропонувано новий метод, заснований на аналізі трафіку. Запропонований метод використовував набір даних CICAndMal2017, який містить 84 функції для чотирьох типів шкідливих програм, включаючи десять типів атак програм-вимагачів. Методи попередньої обробки були застосовані для нормалізації та надмірної вибірки безпечного трафіку, що призвело до набору даних із 402834 екземплярів. Після цього основним кроком у цьому дослідженні була застосована оптимізація рою частинок (PSO) для вибору характеристик трафіку для двійкової (26) і багатокласової (23) класифікації. Було застосовано, проаналізовано та порівняно численні експерименти з використанням класифікаторів DT і RF і різних підмножин вибраних функцій для виявлення на двох рівнях: двійкова класифікація як програми-вимагачі або доброякісна або багатокласова класифікація десяти типів трафіку програм-вимагачів. RF продемонстрував найкращу продуктивність у виявленні програм-вимагачів з точністю 81,58%, тоді як DT було визнано найкращим для багатокласової класифікації.

Нарешті, лише одне дослідження використовувало DL, використовуючи той самий набір даних, що в [41]. В [42] запропонувано метод на основі DL для виявлення програм-вимагачів у середовищі Android за допомогою моделі LSTM. На етапі попередньої обробки було застосовано вісім методів вибору ознак за допомогою інструменту WEKA, і було вибрано 19 найбільш прогнозованих ознак. Збалансовані 40000 зразків (20000: доброякісні, 20000: програми-вимагачі) були розглянуті для експерименту. Вони розділили дані на набори для навчання та тестування 80:20. Потім для

оцінки ефективності моделі використовували численні оціночні метрики. Запропонована модель досягла точності 97,08% і, як стверджується, здатна, масштабована та здатна виявляти атаки програм-вимагачів, якщо їх розмістити в ядрі ОС Android.

На основі розглянутої літератури ми виявили, що деякі обмеження були більш поширеними та частими, ніж інші. Наприклад, 71% досліджень мали проблему невеликого розміру вибірки [4, 25, 36, 37]. Усі ці дослідження мали розмір вибірки менше 5 К, що, як наслідок, призвело до застосування методів ML замість DL, оскільки кількість зразків недостатня для створення моделі DL. Ми можемо зауважити, що з усіх досліджень лише одне дослідження використовувало DL [42]. Лише в одному дослідженні також використовувався неконтрольований ML [40]. Покладення на те, що постачальники антивірусних програм надають явні мітки дослідникам, може бути проблемою, особливо якщо розглядати застосування рішень у реальному часі для виявлення атак програм-вимагачів Android. Таким чином, залежність від історичних прикладів може призвести до поганої роботи моделей під час впровадження невідомої атаки, яку можна вирішити за допомогою неконтрольованого навчання. Незважаючи на відмінну продуктивність, яку забезпечують моделі DL і методи без нагляду, вони недостатньо досліджені в останній літературі. Таким чином, існує відкрита область для дослідження застосування DL та неконтрольованих рішень для виявлення програм-вимагачів Android. Хоча ML може бути життєздатним рішенням у багатьох випадках, розмір набору даних є критичним фактором. У цьому контексті вивчення потенціалу DL стає особливо актуальним через його здатність обробляти складні шаблони та зв’язки в даних, що може призвести до більш надійних рішень для виявлення програм-вимагачів Android, які заслуговують на особливу увагу, оскільки моделі DL відомі своєю масштабованістю. Ось чому вони можуть обробляти великі набори даних. Крім того, прокляття розмірності було загальним обмеженням для більшості досліджень [27-32]. Ці дослідження включали 50

або більше функцій у свої експерименти, які можуть бути дорогими з обчислювальної точки зору. Дані великої розмірності можуть призвести до збільшення складності використовуваних моделей, а також ускладнити інтерпретацію результатів. Крім того, моделі, побудовані з великомірними даними, мають вищий ризик переобладнання, тому вони менш надійні. Деякі дослідження застосовували вибір ознак, але кількість ознак залишалася високою. Крім того, багато досліджень [28-33] мають незбалансований розподіл міток класів, що часто зустрічається в задачах виявлення аномалій. Випадків «аномалії» набагато менше, ніж звичайних/нормальних випадків. Однак це питання має бути вирішено шляхом меншої вибірки для більшості, щоб зрівнятися з меншістю. Незважаючи на втрату цінних даних, це рішення підвищує продуктивність і надійність моделей, оскільки незбалансовані дані можуть призвести до переобладнання або зсуву. Примітно, що ми бачимо, що майже всі дослідження працювали над бінарною класифікацією замість багатокласової класифікації, за винятком [41], де було зроблено обидва. Двійкова класифікація є перевагою, оскільки двійкові класифікатори мають вищі швидкості збіжності. Крім того, небагато досліджень досліджували потенціал аналізу трафіку [34, 36], незважаючи на його багатообіцяючі результати.

У цій роботі ми прагнемо усунути ці обмеження, використовуючи великий набір даних на основі аналізу трафіку для вирішення проблем дисбалансу даних. Ми заповнимо прогалину в літературі, розгорнувши модель ансамблю та дві моделі DL, які не були розгорнуті в існуючій літературі, на додаток до інших моделей ML. Для подолання прокляття розмірності та зменшення складності обчислень слід використовувати менший набір функцій. Це забезпечить передове рішення для виявлення програм-вимагачів Android.

# 2 МЕТОДОЛОГІЯ ДОСЛІДЖЕНЬ

## 2.1 Матеріали та методи

Щоб усунути обмеження наявної літератури та заповнити прогалину, це дослідження вивчило та класифікувало мережевий трафік і класифікувало його як програмне забезпечення-вимагач або безпечне за допомогою нещодавнього набору даних. Корисні атрибути були витягнуті з набору даних для досягнення найкращої продуктивності. Це дослідження було проведено з використанням узгодженої методології, яка включала наступні основні етапи: збір даних, попередню обробку, класифікацію та оцінку. Коли ми отримали набір даних, ми спочатку візуалізували та оцінили дані, щоб зрозуміти наявні функції. Після цього етапи попередньої обробки для покращення якості даних і підвищення продуктивності та надійності моделей були зрозумілі. Етапи попередньої обробки включали двійкову передачу міток, недостатню вибірку, перетворення категоріальних ознак у числові та вибір ознак за допомогою методів прямого вибору ознак і важливості ознак. Потім набір даних було розділено на набори для навчання та тестування 80:20 відповідно, щоб підготувати його до класифікації за допомогою моделей DT, SVM, KNN, ансамблю (DT, SVM, KNN), FNN і TabNet. Нарешті результати були отримані та проаналізовані. На рисунку 2.1 представлено візуальне представлення методологічних кроків, прийнятих для цього дослідження.

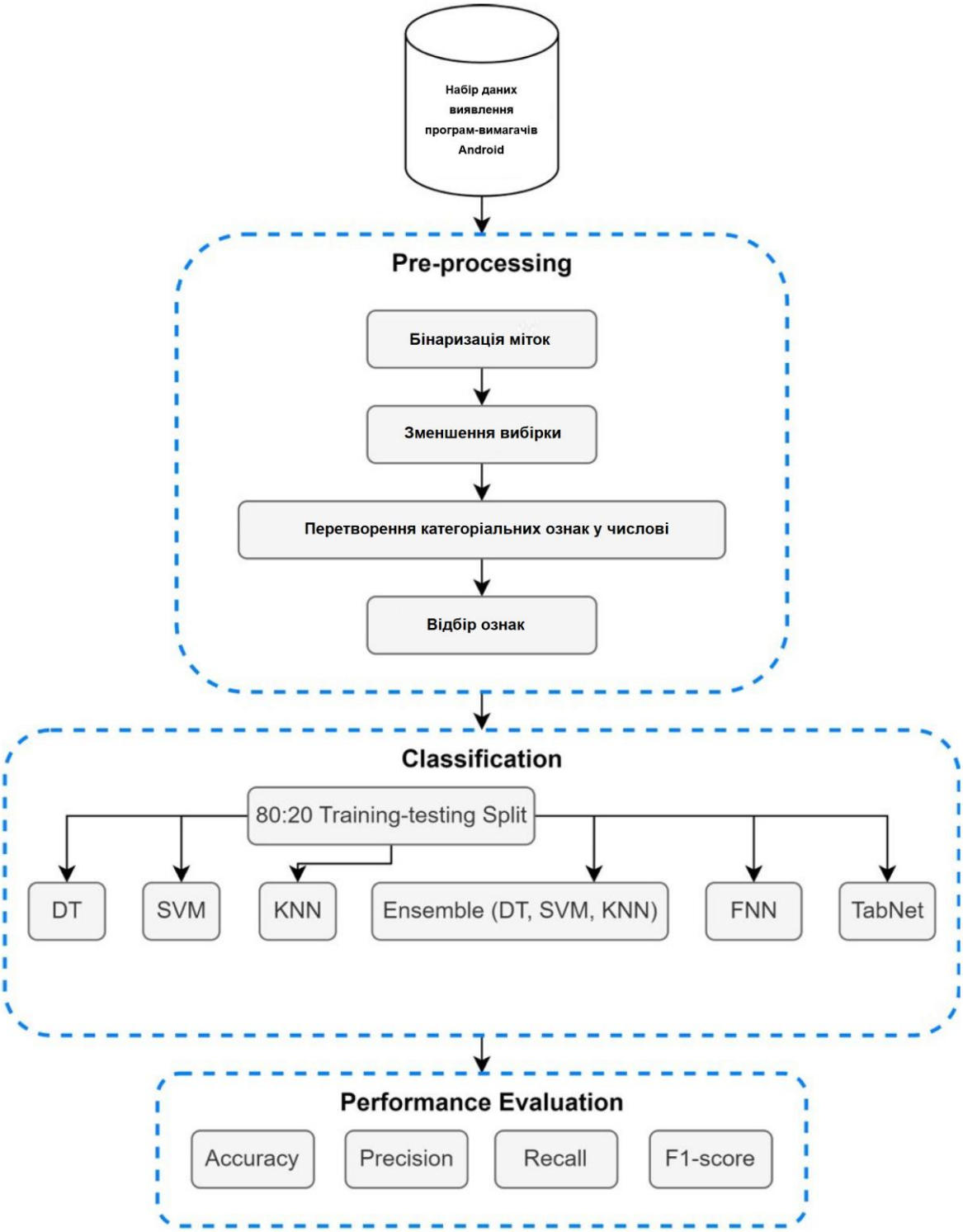


Рисунок 2.1 – Пропонована система методології

## 2.2 Опис набору даних

Набір даних, використаний у цьому дослідженні, є загальнодоступним набором даних, доступним на Kaggle, під назвою Android Ransomware

Detection, який був зібраний і опублікований Субхадіпом Чакраборті з Канадського інституту кібербезпеки (CIC) для виявлення атак програм-вимагачів у мережах Android за допомогою методів машинного навчання (посилання на набір даних). додано нижче в розділі «Доступність набору даних»). Набір даних нещодавній, і, наскільки нам відомо, він раніше не використовувався. Таким чином, немає жодної контрольної роботи над набором даних, яку слід враховувати. Набір даних містить 392035 записів і 85 атрибутів (включаючи атрибут класу). Крім того, набір даних не містить відсутніх значень. Набір даних містить 10 типів програм-вимагачів Android і нешкідливий трафік (загалом 11 міток), як показано на рисунку 2.2. Рисунок 2.2 також демонструє високий дисбаланс у розподілі міток класів у наборі даних. З 392035 записів лише 43091 є безпечним трафіком. Проблема дисбалансу надзвичайно поширена під час вирішення проблем виявлення аномалій. Крім того, у таблиці 2.1 показано всі функції, присутні в наборі даних.

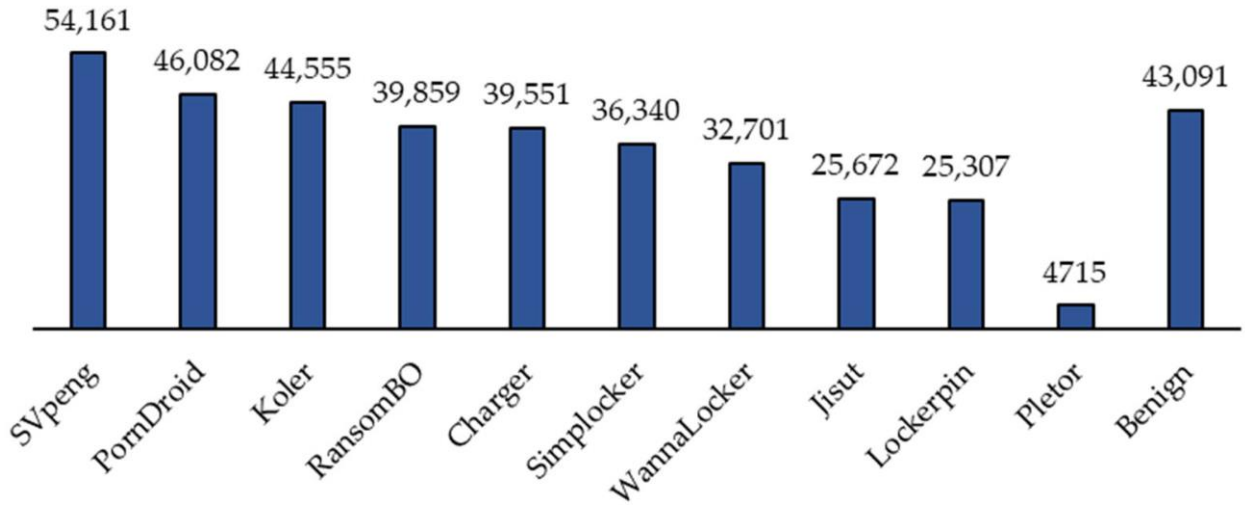


Рисунок 2.2 – Розподіл міток класів

Таблиця 2.1 – Функції, знайдені в наборі даних

| № | Особливість | № | Особливість | № | Особливість |
|---|---|---|---|---|---|
| 1. | Flow ID | 15. | Fwd packet length std | 29. | Fwd IAT max |
| 2. | Source IP | 16. | Bwd packet length max | 30. | Fwd IAT min |
| 3. | Source port | 17. | Bwd packet length min | 31. | Bwd IAT total |
| 4. | Destination IP | 18. | Bwd packet length mean | 32. | Bwd IAT mean |
| 5. | Destination port | 19. | Bwd packet length std | 33. | Bwd IAT std |
| 6. | Protocol | 20. | Flow bytes/s | 34. | Bwd IAT max |
| 7. | Flow duration | 21. | Flow packets/s | 35. | Bwd IAT min |
| 8. | Total fwd packets | 22. | Flow IAT mean | 36. | Fwd PSH flags |
| 9. | Total backward packets | 23. | Flow IAT std | 37. | Fwd header length |
| 10 | Total length of fwd packets | 24. | Flow IAT max | 38. | Bwd header length |
| 11 | Total length of bwd packets | 25. | Flow IAT min | 39. | Fwd packets/s |
| 12 | Fwd packet length max | 26. | Fwd IAT total | 40. | Bwd packets/s |
| 13 | Fwd packet length min | 27. | Fwd IAT mean | 41. | Min packet length |
| 14 | Fwd packet length mean | 28. | Fwd IAT std | 42. | Max packet length |

Продовження таблиці 2.1

| № | Особливість | № | Особливість | № | Особливість |
|---|---|---|---|---|---|
| 43 | Packet length mean | 57. | Subflow fwd bytes | 71. | Idle min |
| 44 | Packet length std | 58. | Subflow bwd packets | 72. | Bwd PSH flags |
| 45 | Packet length variance | 59. | Subflow bwd bytes | 73. | Fwd URG flags |
| 46 | FIN flag count | 60. | Init_Win_bytes_forward | 74. | Bwd URG flags |
| 47 | SYN flag count | 61. | Init_Win_bytes_backward | 75. | RST flag count |
| 48 | PSH flag count | 62. | act_data_pkt_fwd | 76. | CWE flag count |
| 49 | ACK flag count | 63. | min_seg_size_forward | 77. | ECE flag count |
| 50 | URG flag count | 64. | Active mean | 78. | Fwd avg bytes/bulk |
| 51 | Down/up ratio | 65. | Active std | 79. | Fwd avg packets/bulk |
| 52 | Average packet size | 66. | Active max | 80. | Fwd avg bulk rate |
| 53 | Avg fwd segment size | 67. | Active min | 81. | Bwd avg bytes/bulk |
| 54 | Avg bwd segment size | 68. | Idle mean | 82. | Bwd avg packets/bulk |
| 55 | Fwd header length | 69. | Idle std | 83. | Bwd avg bulk rate |
| 56 | Subflow fwd | 70. | Idle max | 84. | Time stamp |

## 2.3 Фаза попередньої обробки

На цьому етапі спочатку була виконана візуалізація даних. Спочатку ми перевірили відсутність значень, розподіл класів, тип атрибутів (категоріальний/числовий) і коефіцієнти кореляції. Було помічено, що 12 атрибутів мали значення 0 для всіх зразків у наборі даних. Тому їх видалили. Це пов’язано з тим, що атрибут із абсолютно однаковим значенням для всіх зразків (нульовим чи ненульовим) не сприяє процесу навчання моделі та може навіть заплутати її. Було видалено наступні функції: прапори Bwd PSH, прапори Fwd URG, прапори Bwd URG, кількість прапорів RST, кількість прапорів CWE, кількість прапорів ECE, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, Bwd Avg Bulk Rate. Крім того, функцію «Time Stamp» було важко перетворити на числовий об’єкт або об’єкт дати й часу, що ускладнювало її обробку. Таким чином, його було видалено. Загалом із зазначених вище причин було видалено 13 функцій. Крім того, було виконано двійкове переведення міток, недостатню вибірку та перетворення категоріальних атрибутів у числові атрибути, що має бути детально пояснено в підрозділах нижче.

### 2.3.1 Бінаризувати мітки

Спочатку набір даних складався з 11 міток класів, включаючи 10 типів атак програм-вимагачів і безпечний клас. Однак у цьому дослідженні ми прагнемо класифікувати зразки мереж як програми-вимагачі або безпечні, незалежно від типу програм-вимагачів. Тому ми перетворили багатокласові мітки лише на (вимагач, доброякісне програмне забезпечення). Клас програм-вимагачів був представлений як 1, тоді як безпечний клас був представлений як 0.

### 2.3.2 Зменьшення вибірки

Після подвійної перетворення міток, щоб усунути дисбаланс класів у наборі даних, ми застосували метод рандомізованої недостатньої вибірки. Цей метод зазвичай використовується для вирівнювання міток класів у наборі даних шляхом зменшення кількості значно більш поширеного класу (програм-вимагачів) у нашому випадку до рівня меншості класу (доброякісного). Таким чином, вибірка обох міток класу була недостатньою, щоб мати однаковий розмір (43 091: програми-вимагачі; 43 091: доброякісні), як показано на рисунку 2.3.

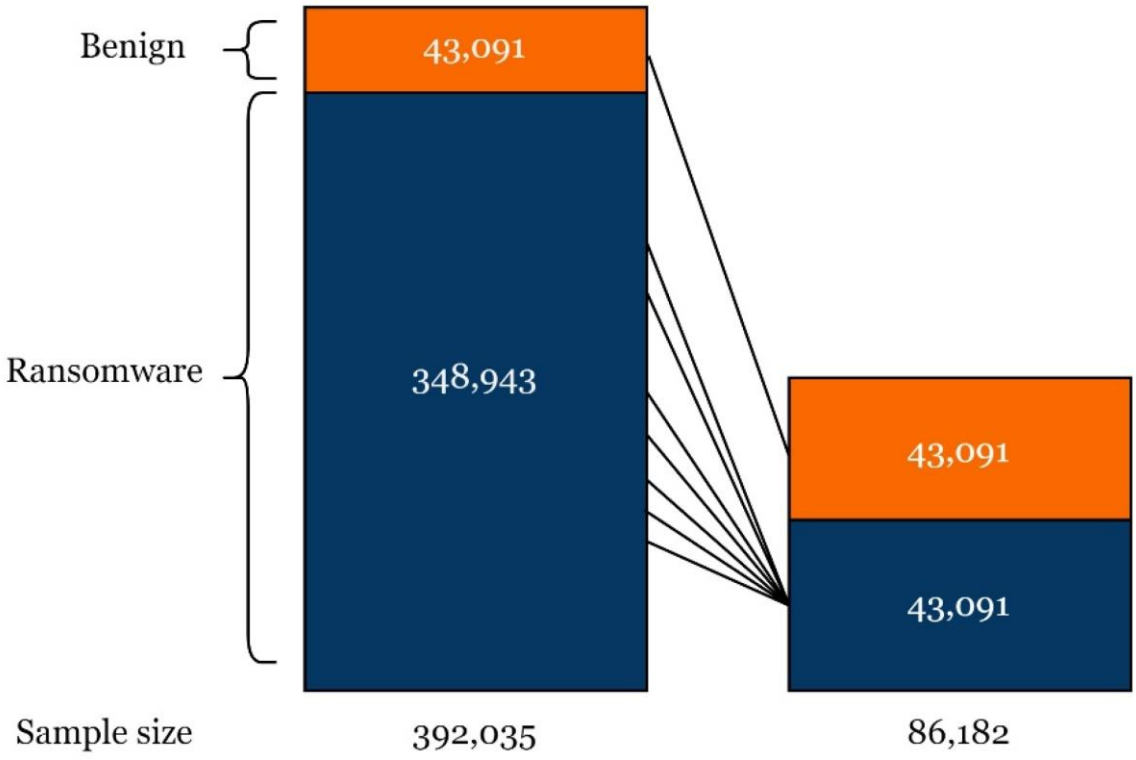


Рисунок 2.3 – Техніка зменьшення вибірки, застосована до набору даних

### 2.3.3 Перетворення категоріальних атрибутів у числові атрибути

Числове представлення категоріальних атрибутів забезпечує послідовний і стандартизований спосіб представлення даних. Це забезпечує

більш ефективні порівняння, обчислення та перетворення, які є важливими для багатьох методів машинного навчання. Таким чином, три категорійні ознаки, а саме IP-адресу джерела, IP-адресу призначення та ідентифікатор потоку, були перетворені в числові.

### 2.3.4 Вибір ознак

Вибір ознак є важливим кроком у ML, який передбачає ідентифікацію та вибір підмножини відповідних функцій із доступного набору вхідних змінних. Це забезпечує покращену продуктивність моделі, можливість інтерпретації, ефективність і можливість узагальнення. У цьому дослідженні відбір ознак виконувався за допомогою двох методів: прямого вибору ознак і важливості ознак (k = 10). Прямий вибір функції – це техніка вибору функції, яка починається з порожнього набору функцій і ітеративно додає одну функцію за раз, щоб створити найефективнішу підмножину. Процес починається з оцінки продуктивності моделі з використанням кожної окремої функції та вибору тієї, яка дає найвищу продуктивність. У кожній наступній ітерації алгоритм додає одну додаткову функцію, яка забезпечує максимальне покращення продуктивності моделі, доки не буде виконано критерій зупинки. Загалом було отримано 22 ознаки за допомогою техніки прямого вибору ознак. З іншого боку, техніка важливості ознак відноситься до процесу визначення релевантності або внеску кожної функції в модель ML. Це допомагає визначити найважливіші характеристики, які мають найбільший вплив на прогнози моделі. Оцінка важливості функції зазвичай розраховується на основі того, наскільки кожна ознака зменшує домішку або помилку в моделі, коли вона використовується для прийняття рішень про розподіл. Що вищий показник важливості функції, то більший вплив ця функція на створення точних прогнозів. За допомогою методики важливості ознак було отримано 32 ознаки. Після вибору загальних характеристик за допомогою ручного вибору було відібрано загалом 19 характеристик, для

яких продуктивність моделі була найкращою. Остаточний набір вибраних функцій показано нижче в таблиці 2.2.

Таблиця 2.2 – Остаточний набір вибраних функцій

| № | Особливість | № | Особливість | № | Особливість |
|---|---|---|---|---|---|
| 1. | ID потоку | 8. | Макс. довжина Fwd пакету | 15. | Активне середнє |
| 2. | Джерело IP | 9. | Довжина Fwd пакета хв | 16. | Активний станд |
| 3. | Порт джерела | 10. | Довжина пакета Bwd хв | 17. | Активний макс |
| 4. | IP призначення | 11. | Init_Win_bytes_forward | 18. | Бездіяльний означає |
| 5. | Порт призначення | 12. | Init_Win_bytes_backward | 19. | Стандартний режим простою |
| 6. | Протокол | 13. | act_data_pkt_fwd | | |
| 7. | Тривалість потоку | 14. | min_seg_size_forward | | |

## 2.4 Фаза класифікації

ML – це підполе штучного інтелекту, де машини можуть вивчати та розуміти дані без необхідності явного програмування. ML дозволяє автономно розв’язувати широкий спектр проблем, що виникають під час обчислень. Удосконалення алгоритмів ML призвело до розробки алгоритмів DL, які є більш складними та математично складними. ML може бути контрольованим, напівконтрольованим або неконтрольованим. У цьому дослідженні ми зосередилися на керованому ML, оскільки він найкраще підходить для отриманого набору даних. Три моделі ML під наглядом, а

також модель ансамблю та дві моделі DL були навчені та перевірені. Це моделі DT, SVM, KNN, модель ансамблю (DT, SVM, KNN), FNN і TabNet.

### 2.4.1 Дерева рішень

DT – це техніка навчання під наглядом, що характеризується непараметричним підходом, який передбачає побудову ієрархічної структури умов if–else відповідно до властивостей вхідних даних. Він діє як предиктор, позначений як h: X→Y, і працює, переходячи від кореневого вузла дерева до листкового вузла, щоб передбачити мітку, пов’язану з вхідним екземпляром X. На кожному кроці цього шляху наступний дочірній вузол визначається на основі того, як розділено вхідний простір. Цей процес поділу зазвичай спирається на характеристики X або попередньо визначений набір правил для поділу [43].

### 2.4.2 Метод опорних векторів

SVM є ефективним методом навчання під наглядом, який можна використовувати як для програм регресії, так і для класифікації. Вони особливо добре працюють для вирішення складних питань, де існує чітка лінія, що розділяє класи. SVM працює, знаходячи ідеальну гіперплощину, яка максимально розділяє класи в просторі ознак. Під час навчання SVM визначають значимість кожної точки навчальних даних для ілюстрації межі прийняття рішень між двома класами. Як правило, лише частина точок навчання – ті, що розташовані на межі прийняття рішення і відомі як опорні вектори – мають значення для визначення межі прийняття рішення. Щоб обчислити нове положення, обчислюється відстань від кожного опорного вектора. Вибір класифікації робиться [44] на основі важливості опорних векторів і відстаней від них, які були виявлені під час навчання.

### 2.4.3 K-найближчих сусідів

Алгоритм KNN вважається одним із найпростіших алгоритмів ML. Він зазвичай використовується для задач класифікації, але також може бути розширений для задач регресії. Це непараметричний алгоритм, у якому мітка класу більшості визначає мітку класу нової точки даних серед її найближчих 'k' (де k — ціле число) сусідів у просторі ознак. Під час навчання KNN зберігає позначені екземпляри навчальних даних, які служать «знанням» для прогнозування. Він отримує вхідні дані, обчислює відстань і знаходить сусідів. Нарешті, коли ідентифіковано k найближчих сусідів, алгоритм KNN призначає мітку класу новому екземпляру на основі схеми голосування більшості. Мітка класу з найбільшою кількістю сусідів призначається як прогнозований клас для нового екземпляра. У випадку рівності, мітка класу може бути обрана випадковим чином або на основі додаткових правил [44].

### 2.4.4 Класифікатор голосування

Відомо, що ансамблеві моделі поєднують прогнози кількох окремих моделей, щоб зробити більш надійні та точні прогнози. Зазвичай він використовується для завдань класифікації, де кожен окремий класифікатор робить прогнози щодо вхідних даних, а остаточний прогноз визначається шляхом агрегування голосів усіх класифікаторів. Класифікатор голосування можна використовувати з різними типами базових класифікаторів, наприклад DT, SVM, KNN або будь-яким іншим класифікатором, який підтримує багатокласову класифікацію. Він також може обробляти комбінацію класифікаторів з різними алгоритмами або налаштуваннями параметрів.

### 2.4.5 Нейронні мережі прямого зв'язку

ШНМ – це тип ШНМ, натхненний обчислювальною моделлю людського мозку. FNN структуровані як взаємопов’язані шари нейронів.

FNN складаються з вхідного, прихованого та вихідного рівнів. Взаємодія нейронів включає зважені зв’язки та функції активації, що дозволяє мережі моделювати складні відносини. Ця структура підтримує здатність FNN апроксимувати різні функції, надаючи їм можливості універсальної апроксимації функцій [45].

### 2.4.6 TabNet

TabNet – це спеціалізована глибока нейронна мережа для табличних даних, яка вміло справляється із завданнями класифікації та регресії [26]. Його архітектура віддзеркалює DT з низкою підмереж для ієрархічного прийняття рішень. На кожному кроці прийняття рішення TabNet вибірково обробляє підмножини ознак, динамічно визначаючи увагу за допомогою навченого механізму, запозиченого з архітектури трансформатора [26]. У поєднанні з можливостями глибокого навчання TabNet особливо ефективний для вилучення шаблонів і створення точних прогнозів у табличних наборах даних.

## 2.5 Фаза оцінювання

Класифікатори ML і DL можна порівнювати за допомогою різних показників ефективності. У цьому дослідженні використовуються точність, точність, запам’ятовування та оцінка F1 для оцінки та порівняння ефективності розгорнутих моделей. Для оцінки точності класифікатора використовується матриця плутанини, яка містить прогнозовані та фактичні класифікації в чотирифакторній таблиці.

Справжній позитивний результат (TP): правильне прогнозування мережевого трафіку програм-вимагачів Android як програм-вимагачів.

Помилковий результат (FP): неправильне прогнозування мережевого трафіку Android-вимагачів як доброякісного.

Істинно негативний (TN): правильне прогнозування доброякісного мережевого трафіку Android як доброякісного.

Помилково негативний (FN): неправильне прогнозування безпечного мережевого трафіку Android як програми-вимагача.

# 3 ЕКСПЕРЕМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

## 3.1 Експериментальна установка

Експерименти проводилися з використанням Python версії 3.9 на Kaggle з GPU P100. Використовувався пристрій на операційній системі Windows 11 Home з 8 ГБ оперативної пам’яті та процесором i7. Набір даних Android Ransomware Detection, загальнодоступний на Kaggle, використовувався після недостатньої вибірки. Зараз набір даних складався з 43091 програми-вимагача та 43091 доброякісного зразка. Після застосування методів вибору ознак (прямий вибір ознак і важливість ознак з k = 10) було відібрано 19 найкращих спільних ознак для навчання та тестування моделей. Для навчання та тестування було виконано розподіл даних 80:20. Загалом було проведено два експерименти з використанням DT, SVM, KNN, моделі ансамблю (DT, SVM, KNN) із random_state = 42, FNN і TabNet. У експерименті 1 були використані всі 70 функцій. У експерименті 2 було використано 19 найкращих характеристик. Модель DT оптимізували за допомогою налаштування гіперпараметрів, а потім застосували пошук у сітці з перехресною перевіркою. Для моделі ансамблю, по-перше, votingClassifier було імпортовано з sklearn.ensemble. Потім модель ансамблю була побудована та передана до класифікатора голосування. Ми зіткнулися з проблемою великих втрат навчання під час процесу навчання моделі, що вказує на те, що моделі було важко ефективно навчатися з даних навчання та зменшувати розбіжності між прогнозованими результатами та фактичними цільовими значеннями. Ця проблема була очевидною, оскільки вартість втрат постійно зростала з кожною епохою. Щоб вирішити цю проблему, ми застосували методи масштабування функцій до вхідних даних. Масштабування функцій має на меті нормалізувати діапазон вхідних функцій, гарантуючи, що вони мають подібний масштаб. Контейнер моделі

було ініціалізовано як послідовний. Детальні налаштування параметрів, застосовані до всіх моделей, наведено в таблиці 3.1 нижче.

Таблиця 3.1 – Налаштування параметрів, що застосовуються до всіх моделей

| Модель | Параметри | Вибір оптимальних значень |
|---|---|---|
| DT | ‘max_depth’ | [None, 5, 10, 15] |
| | ‘min_samples_split’ | [2, 5, 10] |
| | ‘min_samples_leaf’ | [1, 2, 4] |
| SVM | C | [0.1, 1, 10] |
| | Kernel | [‘linear’, ‘rbf’, ‘poly’] |
| | gamma | [‘scale’, ‘auto’] |
| FNN | Функція активації в прихованих шарах | ReLU with 32 units |
| | Кількість нейронів у вихідному шарі | 1 |
| | Функція активації на вихідному рівні | Sigmoid |
| | Розмір партії | 16 |
| | Кількість епох | 10 |
| | Кількість шарів | 3 |
| TabNet | Кількість кроків прийняття рішення в мережі | 64 |
| | Розмір залучення уваги | 32 |
| | Кількість кроків у блоках уваги та агрегації | 5 |
| | Гамма | 1.3 |
| | Кількість самостійно навчених трансформаторів характеристик | 2 |
| | Кількість трансформаторів загальної функції | 2 |
| | Імпульс для нормалізації партії | 0.02 |
| | Максимальне абсолютне значення | 2.0 |

## 3.2 Результати та подальше обговорення

У двох проведених експериментах було зареєстровано чотири метрики оцінювання – точність, точність, запам’ятовування та оцінка F1 – і моделі оцінювали за ними.

У таблиці 3.2 наведено результати, отримані для моделей DT, SVM, KNN, моделі ансамблю (DT, SVM, KNN) з random_state = 42, моделей FNN і TabNet для експерименту 1 і експерименту 2. В експерименті 1 було використано всі 70 функцій. . У експерименті 2 було використано 19 найкращих характеристик. З точки зору точності, найвища точність 97,24% була досягнута DT, потім Ensemble, FNN, SVM, TabNet і найнижча KNN 88,43%. З точки зору точності, DT досягло найвищого значення, тобто 98,50%, і найнижчого 88,96% за TabNet. Відкликання – це метрика, яка отримує доступ до чутливості моделі шляхом кількісної оцінки здатності моделі правильно ідентифікувати позитивні випадки із загальної кількості фактичних позитивних випадків у наборі даних. Таким чином, пригадування є важливим показником оцінки для нашої конкретної проблеми. Усі моделі показали відкликання понад 97%. SVM досяг видатного відкликання у 100%, а TabNet – найменше, тобто 97,28%. Беручи до уваги показник F1, модель DT досягла найвищого результату 98,45%, а найнижчого – 93,77% від KNN.

Модель DT і FNN показала кращу продуктивність в експерименті 2, тобто після вибору ознак. У випадку SVM можна побачити, що результати, отримані до і після вибору функції, залишилися незмінними. SVM автоматично враховує важливість функції під час створення моделі. Він шукає оптимальну гіперплощину, яка максимізує поділ між точками даних з різних класів. Вищі ваги призначаються ознакам, які більше сприяють розподілу класів. Ця характеристика SVM може зменшити потребу в явному виборі функції. Крім того, інша можливість може бути через прокляття розмірності. SVM зазвичай добре обробляє великі набори даних із численними функціями. Ефективно працює у великих просторах.

Таблиця 3.2 – Результати, отримані після експерименту 1 з використанням усіх 70 функцій і експерименту 2 з використанням 19 найкращих функцій

| Методи | Функції | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|---|
| DT | Всі | 96,89% | 98,29% | 98,22% | 98,25% |
| | 19 функцій | 97,24% | 98,50% | 98,40% | 98,45% |
| SVM | Всі | 89,05% | 89,05% | 100% | 94,21% |
| | 19 функцій | 89,05% | 89,05% | 100% | 94,21% |
| KNN | Всі | 88,79% | 90,49% | 97,68% | 93,95% |
| | 19 функцій | 88,43% | 90,10% | 97,74% | 93,77% |
| Ensemble (DT, SVM, KNN) | Всі | 90,44% | 90,37% | 99,91% | 94,90% |
| | 19 функцій | 90,24% | 90,17% | 99,93% | 94,80% |
| FNN | Всі | 89.09% | 89,12% | 99,95% | 94,22% |
| | 19 функцій | 89.10% | 89.13% | 99.95% | 94.23% |
| TabNet | Всі | 89.04% | 89.05% | 99.99% | 94.20% |
| | 19 функцій | 86.84% | 88.96% | 97.28% | 92.94% |

Він здатний справлятися з властивим шумом і надлишковістю у просторі функцій великої розмірності; тому в таких випадках вибір функцій може бути не дуже критичним. У KNN спостерігалося незначне зниження продуктивності в експерименті 2 після застосування методів вибору функцій. Що стосується моделі ансамблю, незважаючи на те, що вона здатна вивчати та автоматично витягувати найбільш відповідні функції без необхідності явного вибору функцій, продуктивність незначно знизилася. Продуктивність TabNet певною мірою знизилася в експерименті 2 після вибору функції. Для кращої візуалізації та порівняння результатів усіх моделей надано інфографіку, показану на рисунку 3.1.

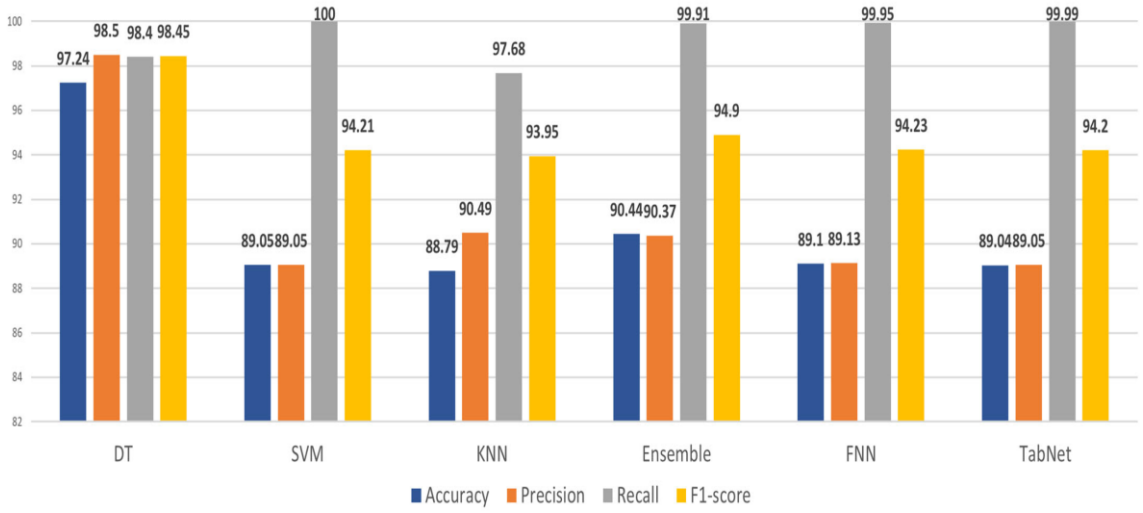


Рисунок 3.1 – Порівняння результатів

Було проаналізовано матриці плутанини для DT, показані на рисунках 3.2 та 3.3. Правильна ідентифікація безпечного мережевого трафіку Android вважається TN; однак правильна ідентифікація трафіку програм-вимагачів Android є TP. Неправильна ідентифікація безпечного мережевого трафіку як програми-вимагача є FN, а неправильна ідентифікація програми-вимагача як безпечного мережевого трафіку є FP. Якщо модель має більше TP і TN (або менше FN і FP), вона вважається більш точною. На рисунках для результатів класифікатора DT ми бачимо, що кількість екземплярів TP та TN більше. Він досяг найвищої точності, точності та оцінки F1. Модель DT ефективна для двійкової класифікації та здатна обробляти великий набір даних числових значень. До оптимізації модель DT досягла точності 97,23%, точності 98,50%, запам’ятовування 98,39% і показника F1 98,44%. Однак завдяки застосуванню налаштування гіперпараметрів за допомогою пошуку по сітці спостерігалися незначні покращення продуктивності моделі. Оптимізована модель DT (з 19 функціями) продемонструвала точність 97,24%, точність 98,50%, запам’ятовування 98,40% і показник F1 98,45%. Ці результати чітко демонструють ефективність налаштування гіперпараметрів у покращенні загальної продуктивності DT.

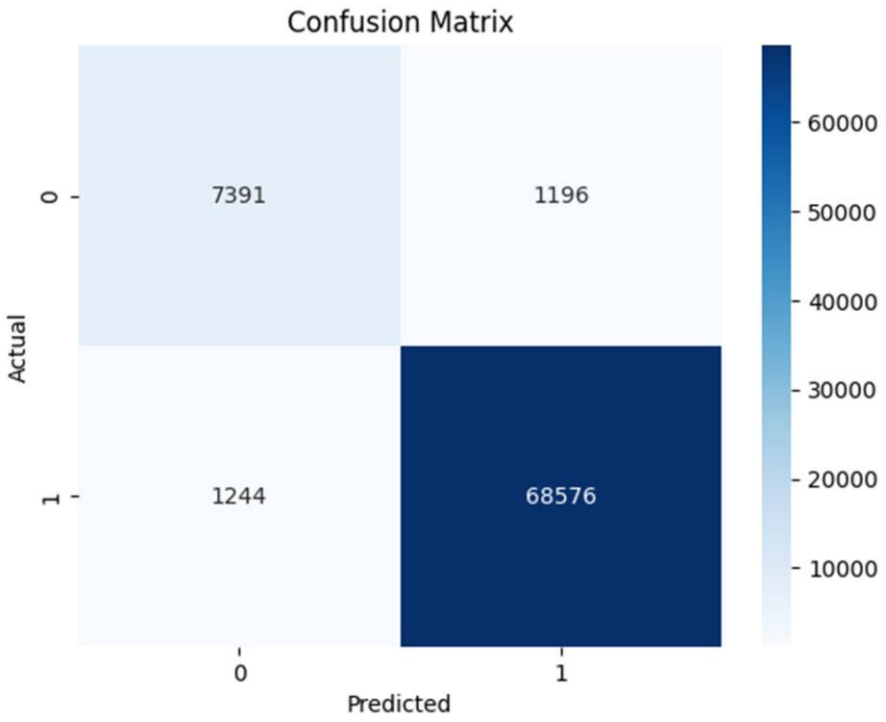


Рисунок 3.2 – Експеримент 1 – матриця плутанини DT

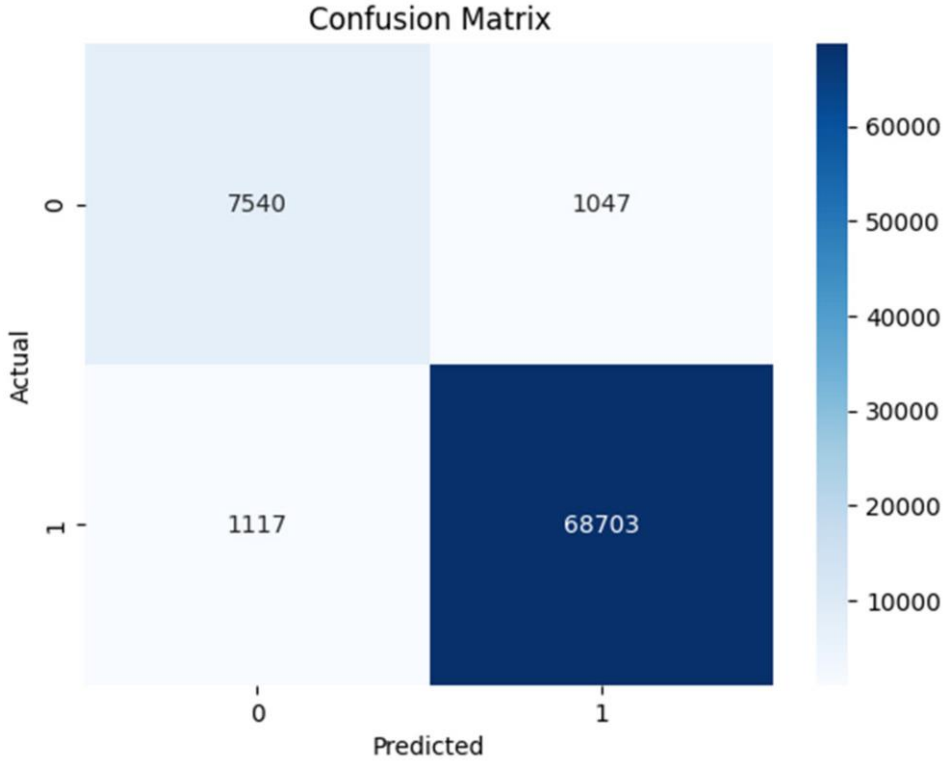


Рисунок 3.3 – Експеримент 2 – матриця плутанини DT

Загалом, DT показав найкращі результати з точки зору точності, точності та оцінки F1, вимагаючи найкоротшого часу виконання 3–4 хв. DT добре вловлює нелінійні зв’язки та обробляє складні взаємодії функцій. Серед моделей ML SVM потребував найдовшого часу виконання 1 год 30 хв, але дав найкраще відкликання 100%. Це можна вважати головним недоліком виявлення атак у реальному часі. Однак SVM є потужним у пошуку оптимальних гіперплощин для розділення класів у просторах великої розмірності. KNN ефективний у визначенні локальних шаблонів і може обробляти різноманітні розподіли даних. В існуючій літературі лише два дослідження використовували моделі ансамблю. В [44] використовували RF модель ансамблю, а в [40] об’єднав BPSO, SVM і SMOTE. В [35] і [40] використовували гібридну модель. J48, NB і LR були об'єднані [35]. RF, J48, LR і RT були об'єднані в [36]. Це показує, що ансамблеві моделі недостатньо досліджено в області виявлення програм-вимагачів Android, що підкреслює необхідність подальшого дослідження потенційних переваг для підвищення продуктивності моделі. Вивчення ансамблевого навчання в цьому контексті виявляється важливим, оскільки дозволяє нам використовувати сильні сторони кількох базових моделей, підвищуючи загальну ефективність системи виявлення програм-вимагачів. Це було основною рушійною силою для проведення експерименту, щоб підкреслити різноманітність серед базових моделей (DT, SVM, KNN) і запровадити жорстке голосування для створення моделі ансамблю. Кожна модель окремо має свої сильні і слабкі сторони. Тому, щоб заповнити наявну прогалину в літературі, ми використали модель ансамблю. Об’єднавши моделі DT, SVM і KNN за допомогою жорсткого голосування, ми побачили, що ми можемо отримати вигоду від їх колективного прийняття рішень і збільшити ймовірність охоплення різних аспектів даних, отримуючи хороші результати. Запропонована модель ансамблю змогла добре працювати на нашому великому наборі даних. Дослідження в літературі, які використовували ансамблеві моделі, досягли точності >99%. Однак вони використовували

невеликі набори даних, які мали проблему дисбалансу даних; отже, існує ймовірність, що ці моделі постраждали від переобладнання. Пригадування є важливим показником оцінки щодо конкретної проблеми, яка розглядається в цьому дослідженні. Запропонована сукупна модель досягла дуже високого показника запам’ятовування 99,93%, маючи лише 19 найкращих характеристик.

Результати, досягнуті двома моделями DL, оцінюються та порівнюються. Було використано FNN, оскільки він більше підходить для типу даних, присутніх у використовуваному наборі даних. Більше того, ми розгорнули його для нашого табличного набору даних, оскільки розмір нашого набору даних був достатньо великим, тобто приблизно 90 K. Він досяг 99,95%, що демонструє, що моделі DL, які не були достатньо досліджені в літературі, здатні показати чудова продуктивність. FNN вважається застарілою та простою трирівневою моделлю, тоді як TabNet розглядається як найсучасніша модель. Тим не менш, FNN продемонстрував кращу точність, точність і показник F1, ніж TabNet. FNN продемонстрував надійну продуктивність із усіма функціями та 19 найкращими функціями. TabNet показав хороші результати з усіма функціями, але з 19 найкращими функціями спостерігалося помітне зниження продуктивності. Порівнюючи природу моделей, модель FNN можна адаптувати за допомогою різних модальностей даних, включаючи зображення, текст і послідовності. З іншого боку, TabNet розроблено спеціально для табличних наборів даних, тому він добре підходить для нашого набору даних. З точки зору часу виконання, FNN вимагав менше 1 години, тоді як TabNet вимагав довшого періоду 7,5 годин. Крім того, завдяки розпаралелюванню коду його вдалося скоротити лише до 4 годин. Враховуючи характер використовуваного набору даних і необхідність отримання своєчасних результатів, FNN стає кращим вибором через його швидке виконання, добре узгоджуючи вимоги щодо швидкої та ефективної роботи в обраному предметі дослідження. Незважаючи на те, що FNN вважається застарілим, його практичність щодо задоволення

конкретних вимог предмета нашого дослідження переважує TabNet. Це підкреслює важливість розгляду релевантності та застосовності моделі над недавністю в певних контекстах.

З точки зору кількісної оцінки, значення FPR для KNN, моделі ансамблю, FNN і TabNet є високими. Ці моделі неправильно прогнозують певну кількість екземплярів програм-вимагачів, якщо ці екземпляри є доброякісними. Проте всі моделі мають надзвичайно низькі значення помилково негативних результатів (FNR), що підкреслює їхню ефективність у виявленні екземплярів програм-вимагачів. Низький FNR має вирішальне значення в контексті безпеки, оскільки пропуск справжньої загрози програми-вимагача може мати серйозні наслідки.

У контексті виявлення програм-вимагачів Android це дослідження в основному було зосереджено на аналізі даних мережевого трафіку. Однак варто відзначити, що пристрої Android оснащені різноманітними вбудованими датчиками, такими як акселерометри, гіроскопи, датчики наближення, мікрофони та датчики температури. Впроваджуючи рішення безпеки для виявлення програм-вимагачів, це дослідження поширюється на захист не лише основних функцій пристрою, а й датчиків Android. Хоча наша поточна робота не передбачала інтеграції даних датчиків, потенційна роль датчиків у покращенні виявлення програм-вимагачів заслуговує на увагу. Датчики можуть надати цінну інформацію про фізичний контекст і взаємодію користувача з пристроєм. На додаток до підвищення точності та адаптивності систем виявлення програм-вимагачів проти нових загроз, поєднання аналізу мережевого трафіку та даних датчиків обіцяє створити динамічний і стійкий механізм захисту.

Нарешті, важливо розглянути порівняльні результати попередніх досліджень. Наприклад, у дослідженні, яке згадується як в [26], DT досяг рівня точності 98,8% із 1923 записами. Навпаки, наше дослідження, у якому використовувався набір даних, що охоплює 392035 записів, досягло високого рівня точності 97,24%. Звертаємо нашу увагу на ансамблеві моделі, в [39]

продемонстровано надзвичайний успіх, досягнувши високої точності 99,67% при використанні набору даних, що складається з 4721 запису. Подібним чином у [40] виняткова точність 99,9% була зареєстрована з використанням набору даних, що складається з 10153 записів. У рамках нашого дослідження, використовуючи значний набір даних із 392035 записів, ми досягли конкурентоспроможного рівня точності 90,4%. Ці порівняльні результати підкреслюють ключову роль, яку відіграє розмір набору даних у впливі на ефективність моделей машинного навчання.

Це дослідження проводилося в симульованому середовищі, а не в реальному світі. Важливо оцінити стійкість моделі в реальному світі проти агресивних атак. Більше того, було брак пояснення отриманих результатів моделей. Такі методи, як локальні інтерпретативні модельно-агностичні пояснення (LIME) або адитивні пояснення Шеплі (SHAP), можуть бути застосовані до пост-хок інтерпретації. Пристрої Android мають обмежені ресурси та обчислювальні можливості, тому моделі ML, призначені для виявлення програм-вимагачів, мають бути легкими та ефективними, щоб працювати на цих пристроях без зниження продуктивності. Потенційним рішенням може бути легке машинне навчання, оскільки воно передбачає створення стислих моделей ML, які підходять для виконання на периферійних пристроях на базі Android. Крім того, отримання відповідних функцій із програм Android може бути складним. Вибір відповідних функцій і їх точне виділення є життєво важливими для продуктивності моделі. Для цього потрібен досвід роботи в галузі та знання характеристик шкідливого програмного забезпечення Android. Оскільки методи вибору функцій можуть знадобитися для зменшення розмірності та видалення нерелевантних функцій, це може вплинути на продуктивність моделі, якщо це зробити не правильно. З точки зору часу, програми-вимагачі постійно розвиваються, регулярно з’являються нові варіанти та методи ухилення.

Модель, навчена на певному наборі зразків програм-вимагачів, може бути неефективною для виявлення нових розробок програм-вимагачів.

Регулярні оновлення та постійне перенавчання моделі необхідні, щоб не відставати від загроз, що розвиваються. Крім того, для забезпечення точності виявлення нових загроз може бути корисним інтегрувати моделі машинного навчання з методами глибокого навчання. Це дозволить покращити адаптивність системи до раніше невідомих типів загроз, зокрема за рахунок аналізу поведінкових патернів.

# ВИСНОВКИ

Останнім часом Android переживає зростання кількості пристроїв, користувачів і технологій, що робить нашу повсякденну діяльність простішою та швидшою. Тим не менш, легкість часто супроводжує незахищеність, що викликає багато проблем щодо конфіденційності та безпеки. Ці проблеми в основному включають кібератаки, які вимагають дуже обережного поводження. Щоб вирішити вищезазначені проблеми, це дослідження намагалося задовольнити поточну потребу та було спрямоване на виявлення атак програм-вимагачів Android за допомогою методів на основі ML та DL. По-перше, було надано комплексний огляд літератури, щоб продемонструвати існуючі дослідження з метою аналізу прогалин і пошуку нових напрямів дослідження. Для проведення двох експериментів було використано останній набір даних виявлення програм-вимагачів Android, 2023 від Kaggle. У рамках попередньої обробки даних для усунення дисбалансу набору даних була прийнята техніка рандомізованої недостатньої вибірки. Після попередньої обробки даних було застосовано вибір ознак за допомогою прямого вибору ознак і важливості ознак. Всього 19 функцій були визнані вирішальними для аналізу та ідентифікації атак. Після вибору функції для навчання та тестування було виконано розділення набору даних 80:20. Було проведено два експерименти з використанням DT, SVM, KNN, моделі ансамблю (DT, SVM, KNN), FNN і TabNet. У експерименті 1 були використані всі 70 функцій. У експерименті 2 було використано 19 найкращих характеристик. Ефективність цих моделей була розрахована з точки зору точності, точності, запам'ятовування та оцінки F1. У результаті DT перевершив результати з точністю 97,24%, точністю 98,50% і результатом F1 98,45%. Найвище відкликання 100% було отримано за допомогою моделі SVM. Також були проаналізовані матриці плутанини для DT. Результати цього дослідження сприяють розвитку галузі виявлення

програм-вимагачів Android, надаючи цінну інформацію та захищаючи пристрої Android перед обличчям нових кіберзагроз.

Крім того, це дослідження підкреслює важливість постійного вдосконалення моделей для адаптації до еволюційних змін у загрозах. Подальші дослідження можуть включати інтеграцію алгоритмів глибокого навчання для покращення виявлення складних атак. Особливу увагу слід приділити оптимізації моделей для роботи в реальному часі, що дозволить забезпечити швидке реагування на кіберзагрози.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Global Mobile OS Market Share 2023|Statista. Available online: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ .

2. Singh, R. An Overview of Android Operating System and Its Security Features. Int. J. Eng. Res. Appl. 2014, 4, 519–521/

3. Ravikumar, J. Cyber Security Threats—Past|Present|Future. August 2017. Available online: https://www.linkedin.com/pulse/cyber-past-present-future-robin-joy/

4. Alsoghyer, S.; Almomani, I. Ransomware Detection System for Android Applications. Electronics 2019, 8, 868.

5. Song, S.; Kim, B.; Lee, S. The Effective Ransomware Prevention Technique Using Process Monitoring on Android Platform. Mobile Inf. Syst. 2016, 2016, 2946735.

6. Ekta; Bansal, U. A Review on Ransomware Attack. In Proceedings of the 2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC), Jalandhar, India, 21–23 May 2021.

7. Number of Ransomware Attacks per Year 2022|Statista. Available online: https://www.statista.com/statistics/494947/ransomware-attacks-per-year-worldwide/.

8. Sharma, S.; Kumar, R.; Krishna, C.R. A survey on analysis and detection of Android ransomware. Concurr. Comput. Pract. Exp. 2021, 33, e6272.

9. Kapratwar, A.; Di Troia, F.; Stamp, M. Static and Dynamic Analysis of Android Malware. In Proceedings of the 3rd International Conference on Information Systems Security and Privacy, Porto, Portugal, 19–21 February 2017.

10. Yunus, Y.K.B.M.; Ngah, S.B. Review of Hybrid Analysis Technique for Malware Detection. IOP Conf. Ser. Mater. Sci. Eng. 2020, 769, 012075.

11. Ляшенко О.С., Великодний І.А., Знайдюк В.Г., Журило О.Д. (2024).

Модель та методи виявлення широкомасштабної атакив середовищі IoT. Системи управління, навігації та зв'язку. 2024. No 1 (75). С 127-132 https://doi.org/10.26906/SUNZ.2024.1.127

12. Aljabri, M.; Mohammad, R.M.A. Click fraud detection for online advertising using machine learning. Egypt. Inform. J. 2023, 24, 341–350.

13. Nagy, N. Phishing URLs Detection Using Sequential and Parallel ML Techniques: Comparative Analysis. Sensors 2023, 23, 3467.

14. Aljabri, M.; Alahmadi, A.A.; Mohammad, R.M.A.; Aboulnour, M.; Alomari, D.M.; Almotiri, S.H. Classification of Firewall Log Data Using Multiclass Machine Learning Models. Electronics 2022, 11, 1851.

15. Aljabri, M.; Zagrouba, R.; Shaahid, A.; Alnasser, F.; Saleh, A.; Alomari, D.M. Machine learning-based social media bot detection: A comprehensive literature review. Soc. Netw. Anal. Min. 2023, 13, 20.

16. Ляшенко О.С., Ступаренко Р.Ю., Знайдюк В.Г. Методи машинного навчання для аналізу трафіку для виявлення програм-вимагачів платформи Android. Проблеми інформатизації. Тези доповідей дванадцятої міжнародної НТК. – Баку: ІСУ АР; Харків: НТУ «ХПІ»; Харків: ХНУРЕ; Харків: НАУ «ХАІ»; Бельсько-Бяла: УТіГН, 2024. – 21-22 листопада 2024. – Том 2. – С. 95.

17. Babbar, H.; Rani, S.; Sah, D.K.; AlQahtani, S.A.; Bashir, A.K. Detection of Android Malware in the Internet of Things through the K-Nearest Neighbor Algorithm. Sensors 2023, 23, 7256.

18. Akhtar, M.S.; Feng, T. Evaluation of Machine Learning Algorithms for Malware Detection. Sensors 2023, 23, 946.

19. Khalid, O. An Insight into the Machine-Learning-Based Fileless Malware Detection. Sensors 2023, 23, 612.

20. Ehsan, A.; Catal, C.; Mishra, A. Detecting Malware by Analyzing App Permissions on Android Platform: A Systematic Literature Review. Sensors 2022, 22, 7928.

21. Kumar, R.; Subbiah, G. Zero-Day Malware Detection and Effective Malware Analysis Using Shapley Ensemble Boosting and Bagging

Approach. Sensors 2022, 22, 2798.

22. Wang, X.; Zhang, L.; Zhao, K.; Ding, X.; Yu, M. MFDroid: A Stacking Ensemble Learning Framework for Android Malware Detection. Sensors 2022, 22, 2597.

23. Alkahtani, H.; Aldhyani, T.H.H. Artificial Intelligence Algorithms for Malware Detection in Android-Operated Mobile Devices. Sensors 2022, 22, 2268.

24. Alraizza, A.; Algarni, A. Ransomware Detection Using Machine Learning: A Survey. Big Data Cogn. Comput. 2023, 7, 143.

25. I. Mykhailichenko, H. Ivashchenko, O. Barkovska and O. Liashenko, "Application of Deep Neural Network for Real-Time Voice Command Recognition," 2022 IEEE 3rd KhPI Week on Advanced Technology (KhPIWeek), Kharkiv, Ukraine, 2022, pp. 1-4, doi: 10.1109/KhPIWeek57572.2022.9916473.

26. Jethva, B.; Traoré, I.; Ghaleb, A.; Ganame, K.; Ahmed, S. Multilayer ransomware detection using grouped registry key operations, file entropy and file signature monitoring. J. Comput. Secur. 2020, 28, 337–373.

27. Arik, S.Ö.; Pfister, T. TabNet: Attentive Interpretable Tabular Learning. Proc. AAAI Conf. Artif. Intell. 2021, 35, 6679–6687.

28. Khammas, B.M. Ransomware Detection using Random Forest Technique. ICT Express 2020, 6, 325–331.

29. Masum, M.; Faruk, M.J.H.; Shahriar, H.; Qian, K.; Lo, D.; Adnan, M.I. Ransomware Classification and Detection with Machine Learning Algorithms. In Proceedings of the 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 26–29 January 2022; pp. 0316–0322.

30. Barkovska O., Mikhal O., Pyvovarova D., Liashenko O., Diachenko V., Volk M. Local Concurrency in Text Block Search Tasks // International Journal of Emerging Trends in Engineering Research, – 2020, – 8(3), – pp. 690-694.

31. Ferrante, A.; Malek, M.; Martinelli, F.; Mercaldo, F.; Milosevic, J. Extinguishing Ransomware—A Hybrid Approach to Android Ransomware Detection; Springer: Berlin/Heidelberg, Germany, 2017; pp. 242–258.

32. Alsoghyer, S.; Almomani, I. On the Effectiveness of Application Permissions for Android Ransomware Detection. In Proceedings of the 2020 6th Conference on Data Science and Machine Learning Applications (CDMA), Riyadh, Saudi Arabia, 4–5 March 2020.

33. Alzahrani, A.; Alshehri, A.; Alshahrani, H.; Alharthi, R.; Fu, H.; Liu, A.; Zhu, Y. RanDroid: Structural Similarity Approach for Detecting Ransomware Applications in Android Platform. In Proceedings of the 2018 IEEE International Conference on Electro/Information Technology (EIT), Rochester, MI, USA, 3–5 May 2018.

34. Abdullah, Z.; Muhadi, F.W.; Saudi, M.M.; Hamid, I.R.A.; Foozy, C.F.M. Android Ransomware Detection Based on Dynamic Obtained Features. In Advances in Intelligent Systems and Computing; Springer: Cham, Switzerland, 2019; pp. 121–129.

35. Almomani, I.; AlKhayer, A.; Ahmed, M. An Efficient Machine Learning-based Approach for Android v.11 Ransomware Detection. In Proceedings of the 2021 1st International Conference on Artificial Intelligence and Data Analytics (CAIDA), Riyadh, Saudi Arabia, 6–7 April 2021.

36. Gera, T.; Singh, J.; Mehbodniya, A.; Webber, J.L.; Shabaz, M.; Thakur, D. Dominant Feature Selection and Machine Learning-Based Hybrid Approach to Analyze Android Ransomware. Secur. Commun. Netw. 2021, 2021, 7035233.

37. Bagui, S.; Woods, T. Machine Learning for Android Ransomware Detection. Int. J. Comput. Sci. Inf. Secur. (IJCSIS) 2021, 19, 29–38.

38. Sharma, S.; Krishna, C.R.; Kumar, R. RansomDroid: Forensic analysis and detection of Android Ransomware using unsupervised machine learning technique. Forensic Sci. Int. Digit. Investig. 2021, 37, 301168.

39. Sharma, S.; Krishna, C.R.; Kumar, R. Android Ransomware Detection using Machine Learning Techniques: A Comparative Analysis on GPU and CPU. In Proceedings of the 2020 21st International Arab Conference on Information Technology (ACIT), Giza, Egypt, 28–30 November 2020.

40. Sharma, S.; Challa, R.K.; Kumar, R. An ensemble-based supervised

machine learning framework for android ransomware detection. Int. Arab. J. Inf. Technol. 2021, 18, 422–429.

41. Almomani, I.; Qaddoura, R.; Habib, M.; Alsoghyer, S.; Khayer, A.A.; Aljarah, I.; Faris, H. Android Ransomware Detection Based on a Hybrid Evolutionary Approach in the Context of Highly Imbalanced Data. IEEE Access 2021, 9, 57674–57691.

42. Hossain, M.S. Android Ransomware Detection From Traffic Analysis Using Metaheuristic Feature Selection. IEEE Access 2022, 10, 128754–128763. [Google Scholar] [CrossRef]

43. Bibi, I.; Akhunzada, A.; Malik, J.; Ahmed, G.; Raza, M. An Effective Android Ransomware Detection Through Multi-Factor Feature Filtration and Recurrent Neural Network. In Proceedings of the 2019 UK/China Emerging Technologies (UCET), Glasgow, UK, 21–22 August 2019. [Google Scholar] [CrossRef]

44. M. Hunko, V. Tkachov, O. Liashenko and J. Rabčan, "Application Architecture For Obtaining Data From Scientometric Databases," 2022 IEEE 3rd KhPI Week on Advanced Technology (KhPIWeek), Kharkiv, Ukraine, 2022, pp. 1-4, doi: 10.1109/KhPIWeek57572.2022.9916398.

45. Shalev-Shwartz, S.; Ben-David, S. Understanding Machine Learning; Cambridge University Press: Cambridge, UK, 2014.

46. Müller, A.C.; Guido, S. Introduction to Machine Learning with Python: A Guide for Data Scientists; O'Reilly: Beijing, China, 2017.

47. Theobald, O. Machine Learning for Absolute Beginners; Independently Published: Chicago, IL, USA, 2018. [Google Scholar]

48. Brownlee, J. Machine Learning Mastery With Python; Machine Learning Mastery: Vermont, Australia, 2016.

49. Ojha, V.K.; Abraham, A.; Snášel, V. Metaheuristic design of feedforward neural networks: A review of two decades of research. Eng. Appl. Artif. Intell. 2017, 60, 97–116.